UNITED STATES AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DIRECT SOLUTIONS OF LARGE, SPARSE LINEAR SYSTEMS

THESIS

GEO/PH/77-3

Michael F. Poore
Captain USAF

14 AFIT/GEO/PH/77-3

1

6

DIRECT SOLUTIONS OF LARGE, SPARSE LINEAR SYSTEMS.

9 Master's THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

16 2304

17 Y1

10 Michael F. Poore B.S.E.E.
Capt USAF
Graduate Electro-Optics

12 153p.

11 December 1977

Approved for public release; distribution unlimited.

1473

012 225

LB

Preface

This report is the summary of my studies in the area of sparse matrices and the results of the programs which I wrote. Although I confined my analyses to Gaussian solution schemes, I wrote the text so that a follow-on student can easily apply some of my recommendations and procedures to other sparse matrix techniques. I also detailed the thinking which I used to build my algorithm; such an algorithm is not widely used for sparse matrix solutions because of limitations of many popular computers. But because of some novel techniques which I used and the strong arithmetic capabilities of the AFIT CDC 6600 Computer, I feel that my algorithm may be of great use to engineers and physicists.

I wish to acknowledge the guidance of my laboratory sponsor, Capt. Carl E. Oliver of the Air Force Weapons Lab, who offered this thesis topic to AFIT and who helped me to clearly define the thesis objectives; one of Capt. Oliver's co-workers, Mr. Mark Gatti, provided excellent and timely support in part of the test phase of this project. I further wish to thank my thesis advisor, Prof. Bernard Kaplan, whose vast experience in Numerical Analysis was a most valuable source in the formulation of my algorithm. Finally, I wish to acknowledge the outstanding performance of my typist, Mrs. Olivia Davis.

Michael F. Poore

Contents

List of Tables

Computer Program Information

The following is a summary of the programs developed by this student as part of this thesis. The algorithms of these programs are directly suited to the CDC 6600 Computer at the Air Force Institute of Technology.

All of the programs are written in CDC FORTRAN Extended, Version 4. Listings of these programs may be obtained from the AFIT Computer Archive, AFIT/AD, Wright-Patterson AFB, OH, 45433.

1. MFP - A Gaussian Elimination sparse matrix solver with various strategic pivoting schemes.

2. MFPTH - A Gaussian Elimination sparse matrix solver with a consecutively calculated pivoting strategy.

3. MFPOP - The same program as MFPTH except that the user can choose either the computed pivoting strategy or an a priori strategy depending on the circumstances of his particular problem.

4. GEBIT - This program represents the same capabilities as MFPOP except that a new packing scheme is used.

5. SMART - The same program as GEBIT except the modular subroutines used in the program development are replaced by program statements within the sparse solver itself. This program is the production model of the Gaussian Elimination algorithm developed in this thesis.

A user's guide to the program SMART can be obtained from the AFIT Physics Department.

Abstract

A comparison is made of the merits of three popular algorithms for direct solutions of large, sparse matrices: Gaussian Elimination, LU Decomposition, and Gauss-Jordan Reduction. The last two algorithms are used in existing sparse matrix solvers at the Air Force Weapons Lab, Kirtland AFB, NM. A mathematical theory discussion explains the algorithms and predicts their performance for arbitrary and strongly structured matrices. The performance comparison involves a wide range of problems practical to technical study at the Weapons Lab. Particular emphasis is placed on solution accuracy and the efficient use of core space. The same test problems are used to analyze the Gaussian Elimination algorithm programmed by this student. From a study of the performance of several Gaussian solution strategies, a new strategy is developed which offers the user a range of options for his particular programming needs. The salient points of this strategy include some stability features of partial pivoting and some array optimization similar to minimum row/minimum column pivoting. The final Gaussian Elimination program is enhanced by a new packing scheme which is highly suited for the CDC 6600 computer: many arrays can be compacted into a single array by subdividing the long computer word structure. A final qualitative comparison is presented from which an optimal solution method is proposed and further study recommended.

DIRECT SOLUTIONS OF LARGE, SPARSE LINEAR SYSTEMS

I. Introduction

Background

Classical Linear Algebra defines a "linear system" as one whose model can be represented by the matrix equation

$$\underline{A}\underline{x} = \underline{b} \qquad (1)$$

where $\underline{A}$ is an "n-by-n" system matrix, $\underline{x}$ is a column vector of solutions, and $\underline{b}$ is a column vector of constants. A "sparse" system is understood to be one whose non-zero elements of the $\underline{A}$ matrix are few: no more than 5% (and typically less than 1%) of the total number of possible entries. Sparse matrices are usually associated with systems whose size (or "rank") is very large (about a thousand).

Large, sparse systems of equations come as a result of work in many fields: physics, engineering, and business management. In technical fields, a frequent use of sparse matrix techniques is in the approximation of the solutions of differential equations. Frequently, the variables of differential equations may not be separable, or the geometry of some problem may not be described by simple, algebraic functions; under these kinds of conditions, classical techniques for solving differential equations cannot be used; an approximation method is necessary.

The following is an illustration of how a system (which happens to have a straightforward analytic solution) can be solved by a finite difference technique which yields a sparse matrix (Ref 11:149, 233-261).

Given: A very long rod whose cross-section is a unit square, and whose heat generation is uniform from within. Problem: To solve for the temperature distribution along the x-axis for the indicated boundary conditions.

Solution: The governing partial differential equation is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 1 = 0 \qquad (2)$$

where "u" is a normalised temperature parameter. The boundary conditions are

$$\begin{aligned} u(1,y) &= 0 & \qquad u(x,1) &= 0 \\ \frac{\partial u}{\partial x}(0,y) &= 0 & \qquad \frac{\partial u}{\partial y}(x,0) &= 0 \end{aligned} \qquad (3)$$

The "exact" analytical solution is

$$2u = 1 + \frac{32}{\pi^3} \cdot \sum_{n=0}^{\infty} (-1)^{n+1} \frac{\cosh[(2n+1)(\pi/2)x]}{(2n+1)^3 \cosh[(2n+1)\pi/2]} \qquad (4)$$

To approximate the partial differential equation, a set of nodal points (Fig. 1) is defined in the region of interest. The distance between adjacent points in the x and y directions are Δx and Δy respectively. The partial differential equation is approximated in the following manner:

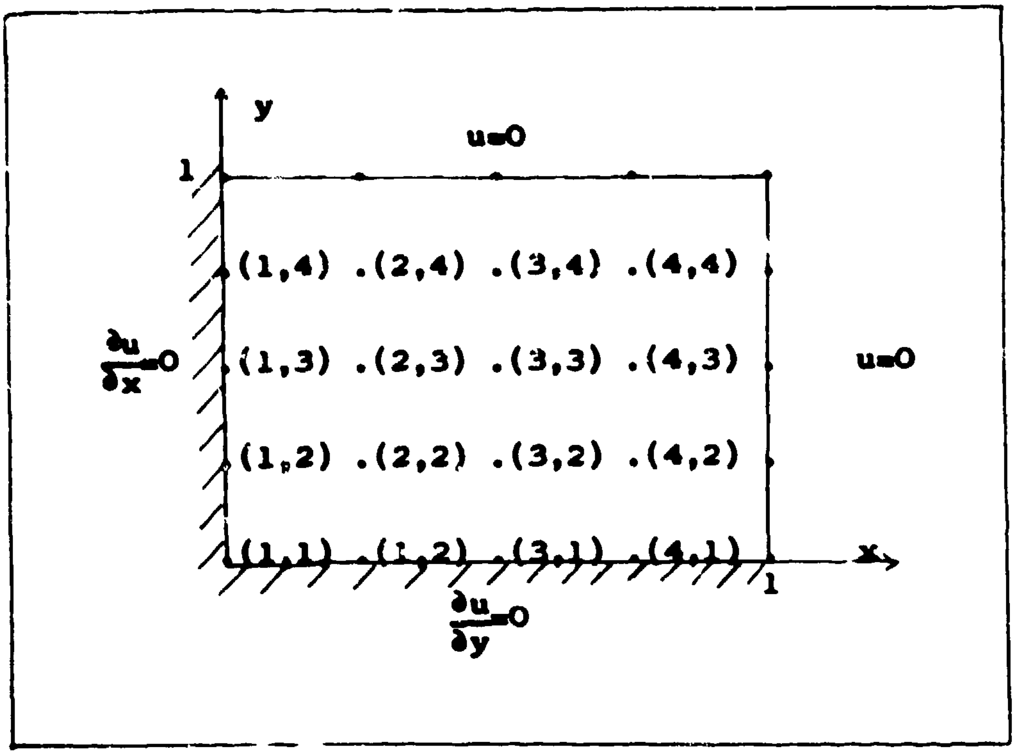


Fig. 1. Nodal Point distribution (From Ref 11:255)

About any node u(m,n),

$$\frac{\partial^2 u}{\partial x^2} \cong \frac{u_{m-1,n} - 2u_{m,n} + u_{m+1,n}}{(\Delta x)^2} \qquad (5)$$

and

$$\frac{\partial^2 u}{\partial y^2} \cong \frac{u_{m,n-1} - 2u_{m,n} + u_{m,n+1}}{(\Delta y)^2} \qquad (6)$$

By substituting Eqs (5) and (6) back into Eq (2), and allowing $\Delta x = \Delta y$, the following equation results:

$$-u_{m-1,n} - u_{m,n-1} + 4u_{m,n} - u_{m,n+1} - u_{m+1,n} = (\Delta x)^2 \qquad (7)$$

By invoking characteristic symmetry of these approximations, and noting that in this case $\Delta x = 1/4$, the sparse system depicted in Fig. 2 is the result.

$$
\begin{bmatrix}
4 & -2 & 0 & 0 & -2 & & & & & & & & & & & \\
-1 & 4 & -1 & 0 & 0 & -2 & & & & & & & & & & \\
0 & -1 & 4 & -1 & 0 & 0 & -2 & & & & & & & & & \\
0 & 0 & -1 & 4 & 0 & 0 & 0 & -2 & & & & & & & & \\
-1 & 0 & 0 & 0 & 4 & -2 & 0 & 0 & -1 & & & & & & & \\
 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & & & & & & \\
 & & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & & & & & \\
 & & & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & & & & \\
 & & & & -1 & 0 & 0 & 0 & 4 & -2 & 0 & 0 & -1 & & & \\
 & & & & & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & & \\
 & & & & & & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & \\
 & & & & & & & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\
 & & & & & & & & -1 & 0 & 0 & 0 & 4 & -2 & 0 & 0 \\
 & & & & & & & & & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\
 & & & & & & & & & & -1 & 0 & 0 & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & 0 & 0 & -1 & 4
\end{bmatrix}
\begin{bmatrix}
u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{1,4} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{2,4} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ u_{3,4} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3} \\ u_{4,4}
\end{bmatrix}
=
\begin{bmatrix}
1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16
\end{bmatrix}
$$

Fig. 2. Sparse Matrix generated from Eq (7) (From Ref 11:255)

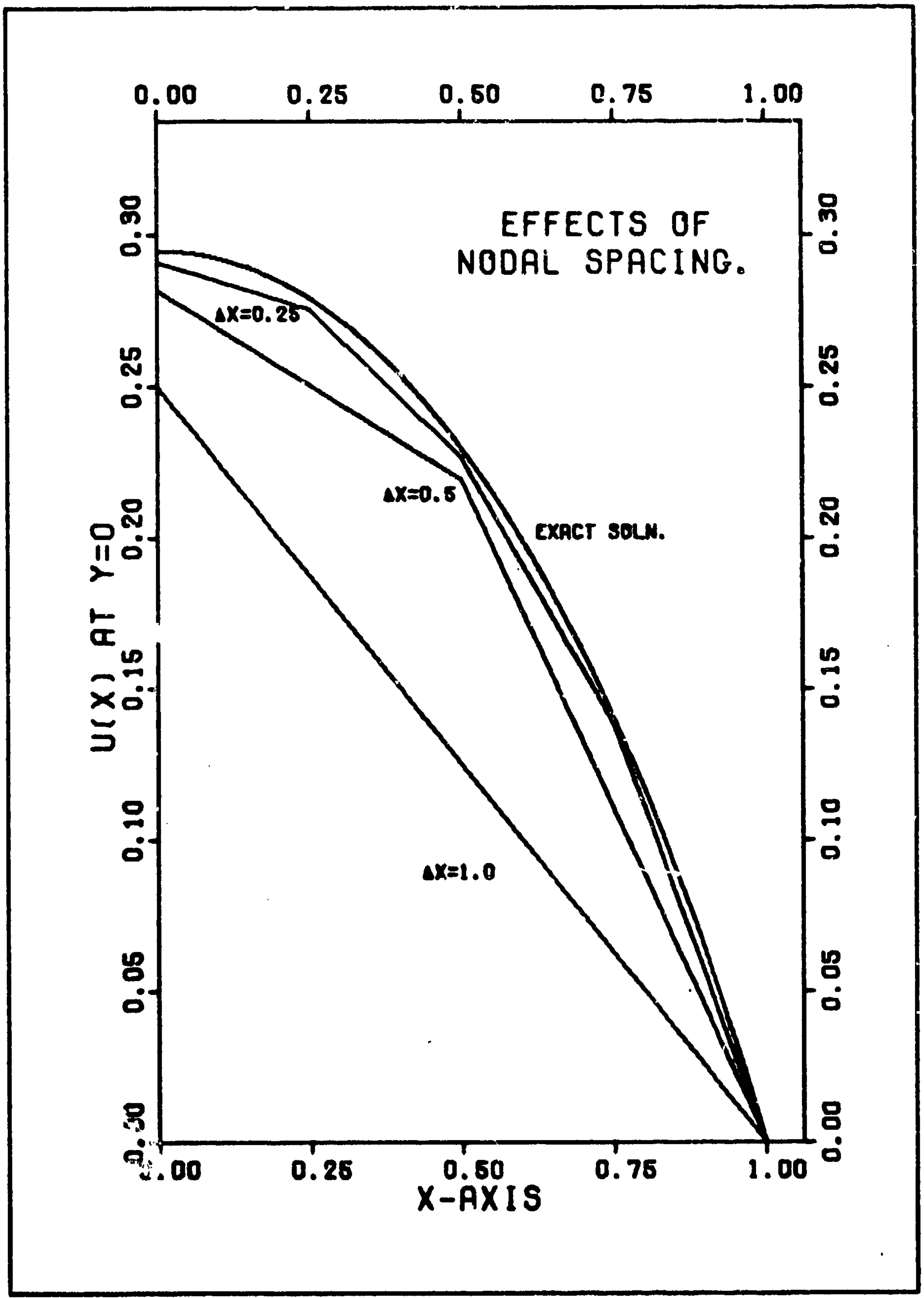


Fig. 3. Effe:ts of Nodal Spacing (From Ref 11:260)

It is important to note that to solve for only four unknown nodes on the x-axis (as the problem stated), it takes 16 equations. Far more rougher approximations yield less accurate results; Fig. 3 demonstrates the effects of an approximation with Δx = 1.0, 0.5, and the chosen 0.25 of this example. Clearly, as Δx gets smaller, the nodal solutions are distributed more closely to the actual analytic solution. As a consequence of shrinking the size of Δx, however, the number of equations increases very quickly. Thus for a near-perfect approximation, a very large number of equations is necessary (hence the development of large, sparse matrices).

In the above example, the curve for Δx = 0.25 is indeed very close to the "exact" solution (Fig. 3); this "good" approximation may suggest that only 16 equations are needed for a reasonably accurate solution (as opposed to the 1000 equations suggested above). However, had the geometry been more arbitrary, the nodal elements of 16 equations would have not provided adequate detail at the boundary. Fig. 4 is proposed as such an example. To attain a good approximation, a vast increase in the nodal density is required. In any case, the use of sparse matrix approximation for the problem of Fig. 4 is much preferred to an analytic solution.

A frequent consequence of sparse matrix construction is that the non-zero elements are distributed in an predominantly diagonal structure with flanking diagonals. The system in Fig. 2 illustrates a tridiagonal core structure with two

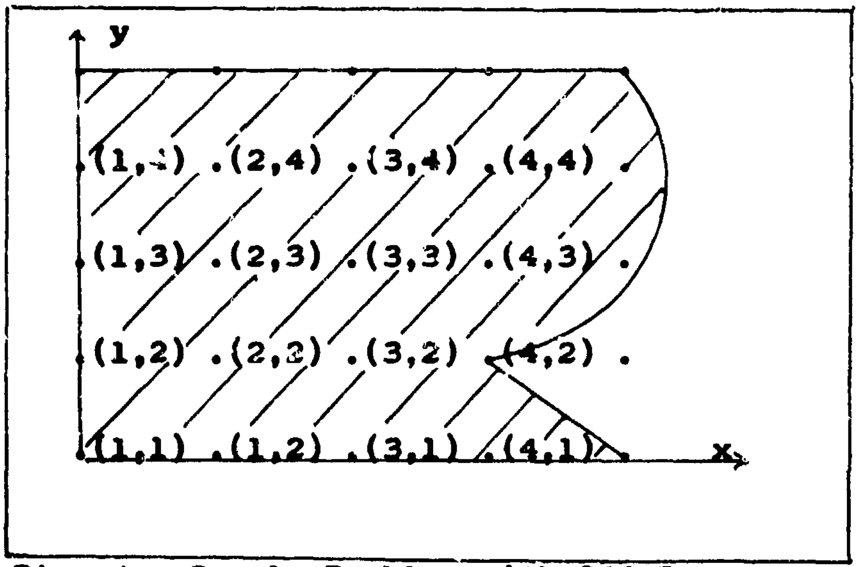


Fig. 4. Sample Problem with Odd Geometry

flanking diagonals each displaced three diagonals away from the main diagonal. These "banded" structures occur often in physics and engineering problems.

<u>Problem</u>

Although linear algebraic equations are theoretically more easy to solve than differential equations, because of the potentially vast number of equations, the system must be solved in a digital computer; the problem is to handle the characteristically large amounts of data as effectively as possible. As an illustration of the problem, a system of size 1000 would require a million locations in the computer core for data alone; since 500,000 locations is a typical upper bound for most large computers, the entire $\underset{\sim}{A}$ matrix could not be stored. Therefore, packing routines must be written which need to store only the non-zero values of $\underset{\sim}{A}$; in this case, a sparse matrix of size 1000 at 5% sparsity would need no more than 50,000 locations to pack the non-zeros.

The actual solution method must also be chosen to maintain sparseness as much as possible as the program runs. A result of the classical inversion solution,

$$\underline{x} = \underline{A}^{-1}\underline{b} \tag{8}$$

is that $\underline{A}^{-1}$ is very dense and will demand core in excess of that available.

Even with an ideal packing scheme, one cannot assume perfect algebraic accuracy in any computer; the conception and growth of errors is a very important consideration in the construction of the sparse matrix solver.

Another factor bearing on the problem is the growth of the $\underline{A}$ matrix as it is computed; new non-zeros (called "fill-in") may be manifested and, in some circumstances, force the data storage requirements beyond the limits allowed.

The pursuit of the solution to this problem is the theme of this thesis.

Thesis Objectives

The following objectives were defined for this project as a result of the motivation of the utility of finite difference techniques and the guidelines of the problem statement:

Comparison of Existing Sparse Matrix Solvers. Two sparse matrix algorithms, already in use at the Air Force Weapons Laboratory, were to be compared. The desired outcome was to find those classes of problems which each algorithm solves the best. The programs compared are The Yale Sparse Matrix Package by Sherman (Ref 14) and a program called "SIMULT" by Key (Ref 9).

A New Packing Scheme. A third sparse matrix solver was programmed as part of this thesis with which the existing programs were compared. A new packing scheme was developed in an effort to exploit the sparseness of the test matrices and more efficiently use the allotted core storage.

The result of the accomplishment of these objectives was a choice from the three programs of the "most desirable" sparse matrix solver as a computational tool.

Standards

There were three significant measures of performance readily available on the computer printouts; one other criterion was rather intangible, but nevertheless, important. The criteria used in judging the sparse matrix solution methods were

1. Accuracy;
2. Core storage requirements;
3. Execution time;
4. The degree that a routine met the user's needs.

The fourth criterion was important in that the outcome of the thesis pertains to engineering problems and not to matrices which are spawned by mere academic curiosity.

The performance of the sparse matrix routine developed as part of this thesis was compared to the existing sparse solvers: the solutions given by Sherman and Key's programs were used as a "performance frame of reference."

Scope

The analyses of this thesis were limited to the performances of the three routines on strictly non-singular, square arrays. The best particular solution for $\underline{x}$ was the goal of each computer program as opposed to the eigenvalue problem.

There are two basic methods of sparse matrix solution: iterative and direct; each of the programs under study was a direct sparse matrix solver. Furthermore, the particular direct methods analyzed were the Gaussian Elimination, the LU Decomposition and the Gauss-Jordan Reduction algorithms. These algorithms were tested against various structures of general sparse matrices; the ramifications of special structures (such as symmetric) were not covered.

Assumptions

The testing of the algorithms against the indicated standards involved practical problems; therefore a certain broad class of problems made up the bulk of the tests. The example of Fig. 1 yielded a "well-conditioned" matrix (defined in Chapter II) which was also diagonally dominant and banded. While it is invalid to assume that all practical sparse matrices are similarly structured, it was assumed that very badly conditioned, near-singular matrices would not generally need to be solved by these programs in practice.

It is also important to assume that the computer into which a future user may load any of these programs is the same make as that used to produce the performance data. Naturally, this assumption implies the ability to duplicate the results of this thesis; but the new packing scheme developed as part of this thesis relies heavily on the word structure of the CDC 6600 Computer (common to both AFIT and AFWL). Any claims for performance based on the experimental data of this thesis must be referred to the hardware superiority which the CDC 6600 computer has over other makes.

Approach

Because of practical limitations, the Yale Sparse Matrix Package program had to be run at Weapons Lab, while SIMULT and the third algorithm were run at AFIT. To the greatest extent practicable, however, the test matrices were standardized so that the results of all three programs were mutually meaningful.

The programming at AFIT was budgeted $700 to complete the project. To efficiently handle the test matrices, the test data were stored on permanent disk files and read into each sparse solver from a local program file instead of from cards. (Naturally, the final production model reads all data from cards.) Some matrix tests were simple, dense matrices with known solutions; these tests were used to verify the operation of the packing schemes and the solution logic.

The standard test matrices were all of size 100; this size was large enough to represent a "sparse" system solution, but not too large as to prohibit execution on the core-limited INTERCOM terminals. (The final variations of the thesis computer work were scaled up to handle larger sizes once the essential comparisons and tests were accomplished.)

Once the basic logic of the new sparse solver was verified, modifications were applied to test various configurations of solution strategy as suggested in the Mathematical Theory (Chapter II). Finally, once an optimimum algorithm was found, a new packing scheme was incorporated as well as other modifications to make the program execute more efficiently.

Thesis Preview

The text of this report contains the mathematical theory required to understand and complete the project, the descriptions of three phases of tests, the method for choosing the optimal program, and a section of conclusions and recommendations suggested by the thesis work.

II. Mathematical Theory

To understand the criteria for choosing the best Gaussian sparse matrix algorithm, it is necessary to first consider the algebraic principles used for linear systems as if they were applied to an ideal problem: "perfect" mathematical accuracy and no computational limitations. The next consideration is the effect of the creation and propagation of errors as the mathematical ideals are constrained by practical limitations. Finally, the scope of the problem to be solved should be considered to decide if a particularly involved solution technique is really required.

The mathematical theory discussed will therefore cover the three principal Gaussian solution methods, the causes of errors, some strategies which attempt to minimize the effects of some errors, and the need for scaling based on the context of the problem to be solved. A summary will include some qualitative predictions for the Gaussian algorithms under comparison.

Gaussian Solutions to Linear Systems

Three algorithms used to solve Eq (1) are the Gaussian Elimination, the LU Decomposition, and the Gauss-Jordan Reduction.

Gaussian Elimination. All of the three solution schemes have their roots in Gaussian Elimination. In the basic form, Gaussian Elimination is a series of n forward operations which transforms $\underline{A}$ into an upper-triangular matrix $\underline{U}$ whose main diagonal elements are unity; then in the back solution, $\underline{x}$ is computed. The terminology used to describe the Gaussian forward process is as follows:

a_{ij} = an original element of $\underline{A}$.

$a_{ij}^{(k)}$ = the value of a_{ij} computed during the k-th operation.

u_{ij} = an element of $\underline{U}$; or, $a_{ij}^{(n)}$.

b_i = an original element of $\underline{b}$.

$b_i^{(k)}$ = the value of b_i computed during the k-th operation.

b_i^* = the final value for b_i.

The forward operation transforms Eq (1) into

$$\underline{Ux} = \underline{b}^* \tag{9}$$

The following is an example of the nomenclature which describes an intermediate step in the forward process:

$$\left[\begin{array}{cc|cccc}
1 & u_{12} & u_{13} & u_{14} & - - - & u_{1n} \\
0 & 1 & u_{23} & u_{24} & - - - & u_{2n} \\
\hline
0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} & - - - & a_{3n}^{(2)} \\
0 & 0 & a_{43}^{(2)} & a_{44}^{(2)} & - - - & a_{4n}^{(2)} \\
\cdot & \cdot & \cdot & \cdot & & \cdot \\
0 & 0 & a_{n3}^{(2)} & a_{n4}^{(2)} & - - - & a_{nn}^{(2)}
\end{array}\right]
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdot \\ x_n \end{bmatrix}
=
\begin{bmatrix} b_1^* \\ b_2^* \\ b_3^{(2)} \\ b_4^{(2)} \\ \cdot \\ b_n^{(2)} \end{bmatrix} \tag{10}$$

The sub-matrix enclosed by the heavy line is referred to as the "k-th derived set"; in the preceding example, the second operation has just been done. The area to the left of the diagonal and to the left of the sub-matrix is strictly zero; the area to the right of the diagonal and above the sub-matrix is the partial set of elements of U. The computer algorithm for the k-th derived set is

$$\begin{array}{rll} a_{ij}^{(k)} = a_{ij}^{(k-1)} - \dfrac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \cdot a_{kj}^{(k-1)} & \left| \begin{array}{l} k=1,2,\ldots n-1 \\ j=k+1,\ldots.n \\ i=k+1,\ldots.n \end{array} \right. & (11) \\ = 1 & \left| \; i=k=1,2,..n \right. & \end{array}$$

and

$$\begin{array}{rll} b_1^{(k)} = b_1^{(k-1)} - \dfrac{b_k^{(k-1)}}{a_{kk}^{(k-1)}} \cdot a_{ik}^{(k-1} & \left| \; i=k+1,\ldots.n \right. & \\ & & (12) \\ = \dfrac{b_i^{(k-1)}}{a_{kk}^{(k-1)}} & \left| \; i=k \right. & \end{array}$$

The term $a_{kk}^{(k-1)}$ is the diagonal or "pivot" element used in the k-th operation. The back solution of Eq (9) is the following computer algorithm:

$$\begin{array}{rll} x_i = b_n^* & \left| \; i=n \right. & \\ & & (13) \\ = b_i^* - \sum_{j=i+1}^{n} u_{ij} \cdot x_j & \left| \; i=n-1,\ldots 1 \right. & \end{array}$$

The preceding applies to a dense matrix; for a sparse matrix, however, for those elements, $a_{ij}^{(k)}$, which are zero, no time-consuming arithmetic operation is necessary. Furthermore, the case may arise in which $a_{ij}^{(k-1)}$ is zero but $a_{ij}^{(k)}$ is computed to be non-zero. This manifestation is called "fill-in." Additionally, a zero may appear on the diagonal; appropriate row or column interchanges can be used to prevent a division by zero. In fact, the proper choice of $a_{kk}^{(k-1)}$ may be dictated by many criteria. The sparse Gaussian Elimination and a study of pivoting strategies is programmed by the student as part of this thesis.

LU Decomposition. The LU Decomposition method makes use of the "LU Theorem" (Ref 7:27) which states that the matrix $\underline{A}$ can be factored into two unique matrices, $\underline{L}$ and $\underline{U}$: $\underline{L}$ is a lower-triangular matrix, and $\underline{U}$ is an upper-triangular matrix whose diagonal elements are unity. The utility of this theorem is that $\underline{L}$ and $\underline{U}$ can be determined without reference to the constant vector, $\underline{b}$. Therefore, once $\underline{A}$ is factored, any set of vectors $\underline{b}$ will yield immediate unique solutions for the corresponding set of $\underline{x}$.

The factorization of $\underline{A}$ into $\underline{L}$ and $\underline{U}$ represents two triangular systems (Ref 7:29):

$$\underline{Ux} = \underline{y} \tag{14}$$

and

$$\underline{Ly} = \underline{b} \tag{15}$$

To use a computer algorithm to factor $\underline{A}$ and to solve Eqs (14) and (15), one can use the following procedure:

$$\underline{Ax} = \underline{b} \quad \text{(Given)} \tag{1}$$

Premultiplication of $\underline{b}$ by a n-by-n identity matrix gives

$$\underline{Ax} = \underline{Ib} \tag{16}$$

Instead of involving $\underline{b}$ in any of the derived sets, as in Eq (12), the algorithm should apply arithmetic operations to the identity matrix: multiplication of a row by a scalar should be carried through the row of $\underline{I}$, and manipulation of elements through row addition should create new elements beyond the diagonal of $\underline{I}$. As a result, the matrix $\underline{I}$ will be transformed into a general matrix $\underline{G}$. It can be proved that $\underline{G}$ is a lower triangular matrix. Therefore, Eq (16) becomes

$$\underline{Ux} = \underline{Gb} \tag{17}$$

Premultiplication of both sides of Eq (17) by $\underline{G}^{-1}$ yields

$$\underline{G}^{-1}\underline{Ux} = (\underline{G}^{-1}\underline{G})\underline{b} \tag{18}$$

which further reduces to

$$\underline{G}^{-1}\underline{Ux} = \underline{Ib} = \underline{b} \tag{19}$$

By uniqueness of the LU Theorem, one therefore concludes that

$$\underline{G}^{-1} = \underline{L} \tag{20}$$

Thus the computer algorithm really solves Eq (16) as

$$\underline{Ux} = \underline{L}^{-1}\underline{b} \tag{21}$$

When $\underline{b}$ is entered into the computation, Eqs (14) and (15) become

$$\underline{Ux} = \underline{y} \tag{22}$$

and

$$\underline{y} = \underline{L}^{-1}\underline{b} \tag{23}$$

Eqs (22) and (23) reduce to a re-statement of Eq (9) since $\underline{y}$ is identical to $\underline{b}^*$. The factorization of $\underline{A}$ and the solution for $\underline{y}$ is the same as the forward Gaussian Elimination process; the back solutions in both Gaussian Elimination and LU Decomposition represent the same procedure.

Any techniques which aid the forward process of Gaussian Elimination (such as row and column interchanges) can be used with LU Decomposition. In theory, therefore, Gaussian Elimination and LU Decomposition give the same results if the same pivoting strategy is used.

In the context of computer operations, he LU Decomposition of $\underline{A}$ can be stored for future use for any number of particular solutions for $\underline{x}$ given any $\underline{b}$. These subsequent solutions represent a considerable savings in computer time. However, extra space must be provided to store $\underline{L}$ as it grows into $\underline{G}$. For a "one-time" solution of Eq (1) the LU Decomposition algorithm may not be appropriate.

The LU Decomposition sparse matrix solver is used by Sherman (Ref 14) in the *Yale Sparse Matrix Package* (YSMP).

Gauss-Jordan Reduction. The Gauss-Jordan Reduction begins with the same matrix setup as in Gaussian Elimination.. The substantial difference is that in the k-th operation, all elements above and below the diagonal (in the k-th column) are eliminated. The computer algorithm for the submatrix of $\underline{A}$ and the computation of the elements of $\underline{b}$ are the same as Eqs (11) and (12) except that the range of the index i is from 1 to n (Ref 13:400,401). The geometry of the k-th derived set, therefore, is not a shrinking square sub-matrix which collapses about the diagonal (Eq [10]) but rather a rectangular sub-matrix whose width collapses from left to right. The following is the nomenclature for the second derived set of Gauss-Jordan Reduction:

$$\left[\begin{array}{cc|cccc} 1 & 0 & a_{13}^{(2)} & a_{14}^{(2)} & - - & a_{1n}^{(2)} \\ 0 & 1 & a_{23}^{(2)} & a_{24}^{(2)} & - - & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} & - - & a_{3n}^{(2)} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ 0 & 0 & a_{n3}^{(2)} & a_{n4}^{(2)} & - - & a_{nn}^{(2)} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \cdot \\ x_n \end{array}\right] = \left[\begin{array}{c} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ \cdot \\ b_n^{(2)} \end{array}\right] \qquad (24)$$

To the left of the dotted line, an identity matrix, $\underline{I}$, is taking shape. Thus, only one forward pass of n operations is needed to solve for $\underline{x}$:

$$\underline{I}\underline{x} = \underline{b}^* \qquad (25)$$

(The constant vector, $\underline{b}^*$, in Eq [25] is not the same final constant vector in Eq [9].) It may first seem that Gauss-Jordan Reduction is the most efficient way to deal with linear systems; however, for a dense matrix, Gauss-Jordan Reduction requires almost 50% more arithmetic operatiohs than Gaussian Elimination (Ref 13:401).

A Gauss-Jordan algorithm usually takes less space in a computer than any Gaussian Elimination program. But in the solution of some problems by Gauss-Jordan Reduction, the exponents of the computed data tend to grow; this growth, in a large system, becomes intolerable even in the best digital computer. Pivoting strategies can be applied to Gauss-Jordan Reduction; however, the effects of a particular pivoting strategy are often different in the Gauss-Jordan algorithm as compared to Gaussian Elimination. John Key's computer program "SIMULT" uses the Gauss-Jordan algorithm (Ref 9).

Errors in Solution Systems

The most important consideration in the solution of a sparse system is that it represents an approximation of some physical system. But to properly analyze the errors spawned in the sparse computer solution, it is assumed that the uncertainties in the given elements a_{ij} and b_j are zero before the operation begins. (The uncertainty of an arbitrary quantity, u, will be annotated as "δu.")

The kinds of errors which have a direct bearing on the solution of sparse systems are round-off error, truncation error, instability, and fill-in proliferation.

Round-off Errors. In a typical digital computer, the product or quotient of a multiplicative operation appears in a double-length accumulator. Before the contents of that accumulator are stored in a data location, the lower order digits are rounded off. For floating point numbers in the CDC 6600 computer, the mantissa of a number can be computed with accuracy up to 14 decimal places provided no other error is introduced.

Truncation Error. Before two numbers can be added in a computer, the smaller number must be right-shifted so that the exponents are normalized. If two numbers whose exponents differed greatly were added, the lower significant digits of the smaller number would be lost; the accuracy once contained in the lost digits would not be carried over into the sum.

One may infer into the discussion of round-off and truncation errors that the smallest algebraic error in a solution scheme may be obtained by minimizing the number of multiplicative and additive operations.

Instability. The "instability" in the solution of a system is a qualitative measure of how algebraic errors have grown to the detriment of the final answer. The following example shows that errors from unstable systems result from the type of algorithm used and not the computer itself.

$$z = \frac{x}{y} \text{ (the algorithm)} \qquad (26)$$

If $x = 1.0$ and $\delta x = 0$, then what are z and δz if there are two values of y and δy ?

$$\begin{aligned} \text{Case I:} \quad & y_1 = 0.0100, \ \delta y_1 = 0.0001 \\ & z = 100, \text{ and } \delta z = 2.0002 \end{aligned} \qquad (27)$$

$$\begin{aligned} \text{Case II:} \quad & y_2 = 1.000, \ \delta y_2 = 0.001 \\ & z = 1, \text{ and } \delta z = 0.002 \end{aligned} \qquad (28)$$

In Case I, z might be stored as 97.9998 (about a 2% error), and in Case II, z might be stored as 0.998 (only a 0.2% error). Even though δy_2 was larger than δy_1, division by the smaller number (y_1) amplified the error much more than division by the larger number (y_2). Even though truncation and round-off errors themselves are on the order of 10^{-14}, if a small number were used as the pivot element, the resulting computation could contain a substantial net error. In this regard, the algorithm of Eq (26) would be deemed relatively unstable if it chose y_1 and relatively stable if it chose y_2. Accordingly, an algorithm which actively seeks the larger numbers for pivot elements is said to be more stable than an algorithm which ignores the relative sizes of possible pivot elements.

Sparse Matrix Fill-in Errors. In a large sparse matrix, it is important to store only the non-zero elements of A; if a fill-in value is calculated, there must be room available to store the new $a_{ij}^{(k)}$. A little fill-in is normally acceptable, but a large amount may exceed the storage capability of a digital computer. More importantly, with the proliferation of fill-in, the algorithm is faced with many more arithmetic operations (resulting in more algebraic uncertainty). Worse yet, in some problems, the fill-in values are relatively small numbers, and the possibility exists that this kind of fill-in may become pivot elements. Further errors due to instability may result. Thus a choice of pivot elements which minimizes fill-in may reduce error growth.

Many pivoting strategies have been developed which attempt to resolve these types of errors.

Pivoting Strategies

A pivoting strategy is a part of a computer algorithm which chooses an element $a_{ij}^{(k-1)}$ to be the new $a_{kk}^{(k-1)}$ (the pivot element) based on some desired outcome. A demonstration for the need for strategic pivoting is found in Appendix A. The following are three examples of the most commonly used strategies for general matrices.

Diagonal Pivoting. Diagonal Pivoting strategy is really no strategy at all. Each of the n operations chooses the diagonal element for the k-th pivot without regard for the results of any previous operation. Hence, instability is possible. Moreover, if a zero were on the diagonal, the computer operation would halt abruptly.

Gaussian Partial Row Pivoting. The Partial Row strategy goes through the rows consecutively; the largest element in the pivot row is selected as the new $a_{kk}^{(k-1)}$. A column interchange in the $\underline{A}$ matrix and a re-arrangement of the components of $\underline{x}$ are necessary to get the pivot element onto the diagonal. The advantages of Partial Row pivoting are that such an algorithm can handle any non-singular matrix, even if a zero were to appear on the diagonal, and that numerical stability is enhanced by use of the largest element. There is, of course, the requirement for extra programming for the column manipulation.

Gaussian Full Pivoting. The Full Pivoting strategy searches the entire submatrix of $\underline{A}$ for the element with the largest absolute value. In this case, a set of row and column interchanges may be necessary. Full Pivoting is considered to be the most accurate pivot scheme for dense matrices; however, not only must additional programming be done for row interchanges, but considerable execution time will be spent searching the remaining submatrix to find the largest value.

The preceding pivoting schemes are those which are classically associated with dense matrices; while pivoting for stability is a good idea, complete disregard for other factors common to sparse matrices can lead to massive errors. Popular sparse pivoting strategies are generally classified as "a priori" or "local" strategies.

A Priori Strategy. An a priori scheme is one in which the overall strategy for the selection of pivoting has been decided for the entire forward process before any operations are done. The most common usage of an a priori pivoting strategy is the case where a system is so vast that it cannot completely reside in the computer core and must be stored on tape or disk. The rows are permuted so that they appear in increasing size. The pivot can be chosen as the first non-zero element of the row (likely to be the diagonal element). A priori schemes can be used for some special cases where the entire array does reside in core.

Local Strategies. A local pivoting strategy checks the present status of the remaining sub-matrix of $\underline{A}$ before the k-th operation; the pivot element is chosen according to the dictates of the strategy. Local strategies are better than a priori strategies in preserving sparsity or operation count (Ref 6:505). The following examples are some of the more popular local strategies.

1. Markowitz's Strategy. The Markowitz procedure chooses the pivot element as that element $a_{ij}^{(k-1)}$ for which the product of the number of non-zeros in the column and the number of non-zeros in the row is a minimum. To scan a large submatrix for the appropriate pivot element would take considerable time. This scheme is meant to minimize the number of arithmetic operations and immediate fill-in. At no time, however, is the absolute value of the pivot considered for numerical stability.

2. Minimum Row/Minimum Column. A scheme which is slightly less effective than the Markowitz strategy but more simple is the Minimum Row/Minimum Column technique. The assignment of $a_{kk}^{(k-1)}$ is given to that element in the smallest column of the smallest row in the remaining submatrix. Again, no checks are made for numerical stability.

3. Minimum Row/Maximum Element. A scheme similar to Gaussian Partial Pivoting, the Minimum Row/Maximum Element technique seeks the largest element of the smallest row in the remaining submatrix. A compromise has been made between the number of computations and stability.

There are many other local pivoting strategies (Ref 4: 92,93) which have been tested; as with these and all of the previously discussed strategies, a dilemma arises. Ideally, it would be desirable to minimize fill-in, maximize stability, and compute the minimum number of calculations as necessary; however, these three criteria are not all mutually exclusive of each other. For example, the Markowitz strategy pivots for computational reduction without regard to stability; Full Pivoting acts to stabilize without regard to the amount of calculation or fill-in. Figure 5 shows a philosophical view of the dilemma. If the "cost" of one criterion were linked with the length of the "line" joining the criterion and the actual strategy used, attempting to shorten one line (to improve performance in that respect) would stretch out the other two, and hence the "cost" would increase. The "cost" would be measured in a rise in computer time or performance degradation.

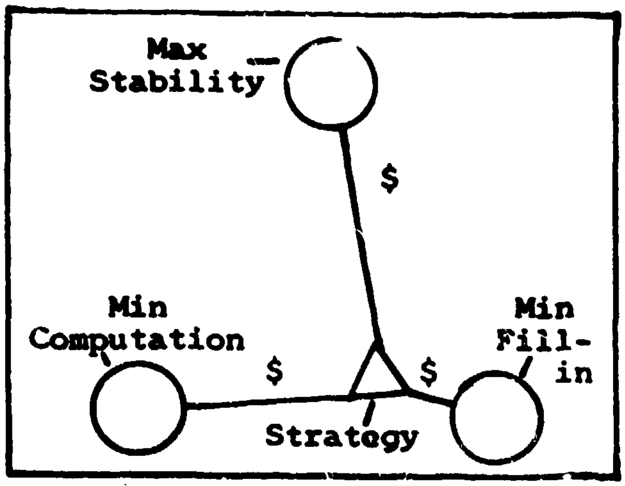


Fig. 5. Pivoting Dilemma

Measuring Strategy Effectiveness. Once any scheme is programmed, it may be of interest to compute the scalar residual error as a performance value. An algorithm would compute this value as the average error per equation, R, in the following manner:

$$R = \frac{1}{n} \sum_{i=1}^{n} \text{abs}(b_i - b_i') \tag{29}$$

where, if x_c is the calculated solution, then

$$b_i' = \sum_{j=1}^{n} a_{ij} \cdot (x_j)_c \tag{30}$$

It is the total effect of the errors which gives rise to R; when discussing the theoretical upper bounds on the errors, it is helpful to define an error matrix, $\delta \underline{A}$ as that matrix which, when added to $\underline{A}$, yields the computed value of $\underline{x}_c$ as if ideal computation took place.

$$(\underline{A} + \delta\underline{A})\underline{x}_c = \underline{b} \tag{31}$$

There is no doubt that, in general, the choice of pivoting strategy has ar important effect on the size of $\delta\underline{A}$ or, more precisely, the Euclidian Norm of $\delta\underline{A}$. The Euclidian Norm of any matrix, $\underline{M}$, is defined as (Ref 13:417):

$$\text{norm}(\underline{M}) = \left[\sum_{i=1}^{n} \sum_{j=1}^{n} m_{ij}^{2}\right]^{1/2} \tag{32}$$

James Bunch adds a new wrinkle to the pivoting dilemma for Gaussian Elimination:

> The error matrix [$\delta\underline{A}$] arising from performing the elimination process in finite precision depends on the fill-in occurring during the elimination. We could seek an ordering of equations so that the bound on [norm ($\delta\underline{A}$)] is minimized. This would <u>not</u> be equivalent to the seeking of an ordering to minimize fill-in. Indeed, we see that minimizing fill-in helps to keep the bound on [norm ($\delta\underline{A}$)] from becoming too large. The problem is even more difficult if we need to pivot for stability (Ref 2:873).

Bunch suggests that the structure of the matrix has a great deal influence on the net error. For example, if the a priori pivoting strategy mentioned on page 24 were used with a tightly banded, diagonally dominant matrix, one would expect very good accuracy and little fill-in. By the structure of the matrix, the chosen pivot element will be from the set of large numbers on the diagonal, and there will be relatively few places within the band structure to allow fill-in. But the same scheme with an arbitrary matrix would not be nearly as successful. In fact, the derivation by Bunch considers the upper bounds for norm ($\delta\underline{\mathbf{A}}$) in the case of banded matrices; for very widely banded matrices or unbanded matrices, the minimization of fill-in may be overshadowed by numerical instability. In any case, the structure of the system to be solved and the desired performance influence the choice of pivoting strategy.

Structure and pivoting have their own peculiar effects on Gauss-Jordan Reduction. The strict error analysis of Gauss-Jordan Reduction is difficult; in Gaussian Elimination, the study of error can be represented by Eq (31) in that the system ($\underline{\mathbf{A}} + \delta\underline{\mathbf{A}}$) represents a "neighboring" system of $\underline{\mathbf{A}}$. In other words, the resulting computed solution, $\underline{\mathbf{x}}_c$, lies in a "neighborhood" of the true solution $\underline{\mathbf{x}}$ as specified by Eq (1). But in Gauss-Jordan Reduction, it is difficult to prove that $\underline{\mathbf{x}}_c$ is always in a neighborhood of $\underline{\mathbf{x}}$ (Ref 12:21); in the context of Eq (31), the system actually solved is not strictly a neighboring system of $\underline{\mathbf{A}}$. The problems associated with Gauss-Jordan Reduction result from failure to control the growth of the elements above the diagonal.

The Gauss-Jordan algorithm can be seen as a combination of above and below diagonal elimination which yields the Identity matrix in Eq (25). The below-diagonal elimination is identical to Gaussian Elimination, and thus the errors from these computations are limited to those which arise from Gaussian Elimination. As for the above-diagonal elimination, no guarantee can be made for any system which ignores stability; but even with Partial Row pivoting, the growth of the above-diagonal elements may be arbitrarily large (Ref 12:21). It is known that positive definite and diagonally dominant $\underline{A}$ matrices are stable with Gauss-Jordan Reduction with Partial Row pivoting; but in the comparison phase of this thesis, it should be emphasized that Key's "SIMULT" program uses Minimum Row/Minimum Column pivoting which is still subject to numerical instability.

Scaling Linear Systems

Algorithms for scaling are used to improve the "condition" of some systems; the relative condition of a system refers to two factors: 1) the relative magnitudes of neighboring elements both before and during elimination, and 2) the uncertainty with which each of the original a_{ij} and b_i were approximated. (Heretofore, δa_{ij} and δb_i were assumed to be zero.) If $\underline{A}$ is "well-conditioned," then the inherent errors $\delta a_{ij}^{(0)}$ and $\delta b_i^{(0)}$ will not be amplified; but in an "ill-conditioned" system even a small error is likely to grow past acceptable limits (Ref 13:396,397).

For example, the situation may arise when b_1 is measured in milliwatts and b_2 is measured in killowatts; the corresponding a_{1j} and a_{2j} will necessarily be out of proportion. No pivoting strategy alone could be stable enough to handle this sort of problem. However, the rows and columns of $\underline{A}$ can be scaled to a more workable size relationship such as a row or column norm (Ref 15:10).

To bring neighboring rows into line, each column should be divided by that column element with the largest absolute value; the matrix which scales $\underline{A}$ this way is a diagonal matrix, $\underline{D}_1$, whose elements are the reciprocals of those maximum column elements. To maintain equivalence of Eq (1), postmultiplication by $\underline{D}^{-1}$ is required:

$$\underline{A}\ \underline{D}_1\ \underline{D}_1^{-1}\underline{x} = \underline{b} \tag{33}$$

Then, to align the columns, a row scaling is required; the row element with the largest absolute value is divided into the row and corresponding b_i. The scaling matrix is another diagonal matrix, $\underline{D}_2$, whose elements are the reciprocals of these row scales. The solution $\underline{x}$ of Eq (1) is the same as that of the following (Ref 15:11):

$$\underline{D}_2\ \underline{A}\ \underline{D}_1\underline{D}_1^{-1}\underline{x} = \underline{D}_2\underline{b} \tag{34}$$

Eq (34) reduces to the final form of

$$\underline{A}'\underline{x}' = \underline{b}' \tag{35}$$

where

$$\underline{x}' = \underline{D}_1^{-1}\underline{x} \tag{36}$$

$$\underline{A}' = \underline{D}_2\underline{A}\underline{D}_1 \tag{37}$$

and

$$\underline{b}' = \underline{D}_2\underline{b} \tag{38}$$

Since $\underline{D}_1$ and $\underline{D}_2$ are diagonal matrices, their storage requirements are only n locations each for the diagonals, and their inverses are easily calculated.

For a program which is designed to be an all-encompassing sparse matrix solver, a scaling algorithm should be an integral part. However, if problems are limited to those with well-conditioned systems, scaling need not be used. In fact, scaling would require extra multiplications for each non-zero, and the computer search for the scaling elements would be time-consuming (even if simple).

Therefore, if some regard is paid to numerical stability in the solution algorithm, and if the scope of the problems to be solved is reasonably constrained, no scaling algorithm is really needed.

Theory Summary and Predictions

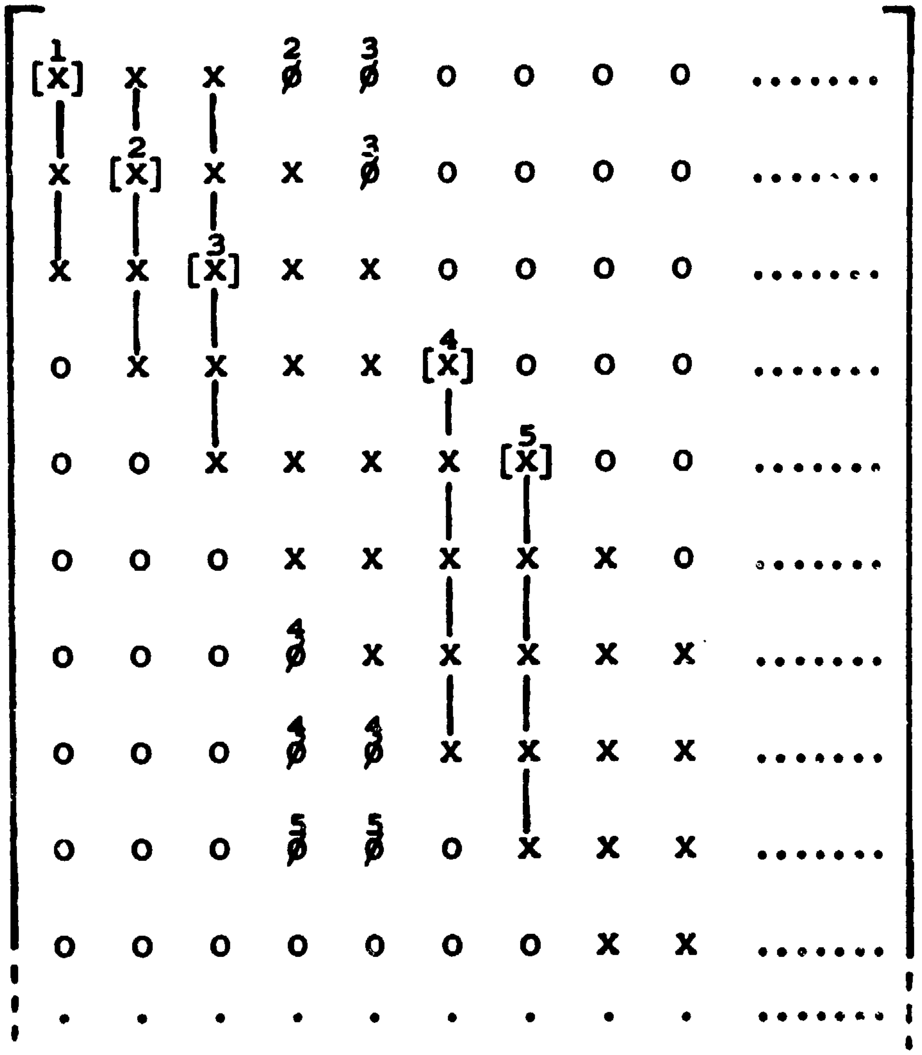
Based on the preceding discussions, some predictions for the Gaussian sparse solvers can be made as a result of the mathematical theory. Short analyses of pentadiagonal and arbitrary matrices will be discussed for Gaussian Elimination and Gauss-Jordan Reduction. (Gaussian Elimination and LU Decomposition will be classified together since they are arithmetically similar.)

Pentadiagonal Case. A diagonally dominant, pentadiagonal matrix is a common problem to solve in nuclear physics. With this structure, almost any a priori or local pivoting strategy in Gaussian Elimination will choose pivots consecutively on the diagonal. The notable exception may be Full Pivoting for which no guarantees can be made. Also, in the Minimum Row/Minimum Column scheme, it is possible that two or more rows have the smallest size (rows one and n, for example); but most algorithms usually have "tie breaking" rules which choose the first element to meet the criteria of the strategy as the pivot. In this pentadiagonal case, there should be no fill-in and the accuracy should be very good.

On the other hand, the Gauss-Jordan Reduction with the Minimum Row/Minimum Column pivoting strategy will fill in greatly with a pentadiagonal matrix. (This strategy is that of the SIMULT program to be compared in this thesis.) The fill-in of at least two values per row (in columns four and five) will occur in all but the first two and last two derived sets (Fig. 6). Furthermore, these fill-in values will have been calculated using previous fill-in. And lastly, the final pivot operations will be in columns four and five and thus are bound to yield significant errors.

Arbitrary Case. Gaussian Elimination will show a wide range of performance with different pivoting schemes. For example, Full Pivoting could easily choose a pivot in the largest row and create vast amounts of fill-in. Even Minimum Row/Minimum Column could "jump" around the matrix for a proper pivot; as a result, even though immediate fill-in is localized and small in amount, it would, in fact, remain to be used repeatedly. Thus fill-in could enter into many calculations and perhaps even become a pivot later on.

Conversely, in Key's Gauss-Jordan Reduction, not only will the immediate fill-in be minimized, but also the fill-in



$$\begin{bmatrix}
\overset{1}{[X]} & X & X & \overset{2}{\emptyset} & \overset{3}{\emptyset} & 0 & 0 & 0 & 0 & \cdots\cdots \\
X & \overset{2}{[X]} & X & X & \overset{3}{\emptyset} & 0 & 0 & 0 & 0 & \cdots\cdots \\
X & X & \overset{3}{[X]} & X & X & 0 & 0 & 0 & 0 & \cdots\cdots \\
0 & X & X & X & X & \overset{4}{[X]} & 0 & 0 & 0 & \cdots\cdots \\
0 & 0 & X & X & X & X & \overset{5}{[X]} & 0 & 0 & \cdots\cdots \\
0 & 0 & 0 & X & X & X & X & X & 0 & \cdots\cdots \\
0 & 0 & 0 & \overset{4}{\emptyset} & X & X & X & X & X & \cdots\cdots \\
0 & 0 & 0 & \overset{4}{\emptyset} & \overset{4}{\emptyset} & X & X & X & X & \cdots\cdots \\
0 & 0 & 0 & \overset{5}{\emptyset} & \overset{5}{\emptyset} & 0 & X & X & X & \cdots\cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & X & X & \cdots\cdots \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdots\cdots
\end{bmatrix}$$

X = location of an original non-zero.

$\overset{k}{[X]}$= the k-th pivot element.

$\overset{k}{\emptyset}$ = a location of fill-in from the k-th operation.

$\begin{matrix} X \\ | \\ X \end{matrix}$ = elements of a column eliminated in the k-th operation.

Fig. 6. Pentadiagonal Fill-in from SIMULT

is likely to be eliminated soon after its creation; thus even with large amounts of fill-in, the errors are less likely to cascade as badly as in Gaussian Elimination.

Therefore, it is predicted that Key's program will outperform Gaussian Elimination on very unstructured matrices, while Gaussian Elimination proves to be more effective on more structured systems. At some "degree" of randomness, both performances should be comparable. Also, in the study of Gaussian Elimination with pivoting, good accuracy may be attained by strategies which either eliminate fill-in shortly after its inception or localize fill-in so that it is not involved with too many subsequent calculations; this claim should hold true even in cases with large amounts of fill-in. Finally, the type of problem which the user has in mind will be the guiding force in choosing the alg rithm and pivot strategy.

III. Comparisons of the Gaussian Solvers

The testing of three major Gaussian algorithms was done on CDC 6600 computers at the Air Force Weapons Lab (AFWL) and the Air Force Institute of Technology (AFIT). Standard main programs were used for all three algorithms to accomplish the following functions: packing the sparse matrix into an appropriate form, executing and timing the particular Gaussian algorithm, computing an average scalar residual error, and printing out the solution.

The comparisons of these programs contain a capsule description of each sparse solver, the initial testing procedure, and a summary which suggests the direction of further study.

Capsule Descriptions of Gaussian Programs

The names of the programs under study are the Yale Sparse Matrix Package (YSMP), by Sherman; the "SIMULT" program, by John E. Key; and the "MFP" study by this student.

YSMP. While Sherman's YSMP program contains many FORTRAN subroutines for solving various special types of sparse matrices (symmetric, for example), only those subroutines needed for general sparse matrices were compared. The YSMP uses an a priori pivoting scheme for LU Decomposition; a permutation array is generated to order both the rows and the columns of A for pivoting (Ref 14:15). The packing scheme is similar to that suggested by Gustavson

(Ref 8:43,44); only the non-zero elements of A are packed, and a row and column pointer table computes the "address" of the desired a_{ij} for future computation.

SIMULT. Key's SIMULT program uses Gauss-Jordan Reduction with Minimum Row/Minimum Column pivoting. The calling program must supply an important data value called "ZTEST"; during the calculations, if the magnitude of a result is computed to be less than ZTEST, it is automatically set to zero. The packing scheme uses a compressed, two-dimensional FORTRAN array for A; the maximum number of allowed non-zeros per row is determined by the user. (Key recommends no more than 20 to 30 elements per row as adequate to handle fill-in.) A similarly structured two-dimensional pointer array stores the "J" column coordinates of the corresponding A values (Ref 9:10).

MFP. The MFP program is a study of Gaussian Elimination with various pivot strategies. The packing scheme is identical to that used in YSMP. The following variations used the indicated pivot strategies in the course of the algorithm construction and the initial testing:

MFP1 - Diagonal
MFP2 - Row Partial Pivoting
MFP3 - Gaussian Full Pivoting
MFP4 - Minimum Row/Minimum Column
MFP5 - Minimum Row/Maximum Element.

Instead of re-shuffling the rows and columns for pivoting, the program stored the order of row and column pivot coordinates into two arrays called IPIV and JPIV. These arrays were then passed to the back solution subroutine to properly compute $\underline{x}$. The resulting matrix $\underline{U}$ may not have appeared to be upper-triangular; but if the row and column interchanges were done as prescribed by IPIV and JPIV, $\underline{U}$ would have indeed appeared as upper-triangular. (Appendix B contains the flow charts for the most important subroutines in MFP.)

Testing Procedures

The criteria for the initial testing of the three Gaussian sparse solvers were the required program space, the orders of magnitude of the scalar residual errors, and the execution time for four standard matrix problems.

Program space. Table I contains a summary of Appendix C; this comparison lists the storage space required for the Gaussian algorithms excluding the main programs and the FORTRAN system routines. All of the variations of MFP are included.

Table I
Essential Programming Space for Sparse Solvers

Program	Length (Octal): Program	Data
YSMP	2134	20207
SIMULT	532	17662
MFP1 - Diagonal	757	16666
MFP2 - Partial	763	16666
MFP3 - Full	1034	16666
MFP4 - Min Row/Min Col.	1335	16666
MFP5 - Min Row/Max Ele.	1145	16666

The data storage requirements were set as that space necessary to solve any 100-by-100 system at 5% sparsity.

Standard Test Results. Four test matrices (Appendix D) were run in each configuration; all tests of YSMP were done at AFWL and tests of SIMULT and MFP were done at AFIT. The average scalar residual error was calculated in each routine as in Eq (29), and the "TIMER" function (Ref 10) provided the time required to execute only that portion of the programs that called the Gaussian solvers. As a result, each Gaussian solver was examined truly independently. The first phase of the comparison is listed in Table II.

Table II
Initial Comparisons of Sparse Solvers

Test Matrix	YSMP	SIMULT*	MFP1	MFP2	MFP3	MFP4	MFP5
#1							
Error:	10^{-14}	10^{+13}	10^{-14}	10^{-14}	10^{-1}	10^{-14}	10^{-14}
Time:	0.12	0.28	0.48	0.48	2.67	1.31	0.58
#2							
Error:	10^{0}	failed	10^{+1}	10^{+1}	10^{+11}	10^{+1}	10^{+1}
Time:	0.11		0.47	0.65	0.93	0.55	0.74
#3							
Error:	10^{-8}	10^{-3}	10^{-8}	10^{-8}	10^{-3}	10^{-8}	10^{-8}
Time:	0.11	0.33	0.47	0.60	1.24	0.55	1.32
#4							
Error:	10^{-14}	10^{+38}	10^{-14}	10^{-14}	10^{-14}	10^{-14}	10^{-14}
Time:	0.06	0.26	0.29	0.29	0.68	0.88	0.37

Error calculated as in Eq (29).
Time measured in seconds.
*ZTEST for SIMULT runs = 10^{-10}.

The first phase of testing confirmed two important aspects of the theory section:

1. LU Decomposition and Gaussian Elimination can yield comparable accuracy.

2. The Gauss-Jordan Reduction in SIMULT performed poorly for very structured matrices.

Test matrix #2 is a near-singular matrix; both YSMP and MFP solved it, although badly. But SIMULT determined the matrix to be singular; this is due to one or more critical elements being computed to be less than ZTEST. As a result, some important non-zero data was cast aside resulting in a singularity. In any case, test matrix #2 was a bad test, and no other conclusions should be drawn from its results.

As predicted, LU Decomposition and Gaussian Elimination always gave the same order of accuracy (except for the Gaussian Full Pivoting). Interestingly, all of the pivot strategies of MFP (except for Full Pivoting) chose pivot elements consecutively on the diagonals. It appears that the YSMP probably chose the diagonal; most a priori strategies would choose the diagonal for a pentadiagonal matrix. Additionally, the actual lues of the errors came very close to those of MFP which did use the diagonal. There is, however, a disparity in the time criterion.

The LU Decomposition should have taken more time than Gaussian Elimination; but a check of the program structures would explain part of this disparity. Most of the repeated operations of the MFP pivot subroutines are contained in other individual subroutines; each call to a FORTRAN subroutine requires more time than a simple "GO TO __" statement. The call to subroutine causes a transfer of control from the calling program to the computer's operating system in order to find the subroutine, execute it, and return to the calling program. The pivot subroutines in MFP must frequently use some external programs called FETCH, DELETE, ROWDIV, and STORE which manipulate data in the compacted form of the sparse matrix (Appendix B). YSMP, on the other hand, is built so that all of the necessary programming for a specified step is contained within the entire subroutine (Ref 14:18):

SORDER - Computes minimum ordering.
NSRORD - Re-orders $\underline{A}$ given the ordering from SORDER.
SSFAC - Computes the symbolic factorization of the re-ordered $\underline{A}$ matrix.
NSNFAC - Computes the numeric factorization of $\underline{A}$, given its symbolic factorization.
NSBSLV - Solves Eq (1) given the LU factorization of $\underline{A}$.

None of these subroutines needs to communicate with any others; they merely must be executed in a prescribed sequence. Finally, an a priori pivot scheme is basically faster than a local pivoter when dealing with pentadiagonal or tridiagonal matrices. Of course, the core usage is larger than MFP because of these speed capabilities.

Study Areas for Next Phase

One of the important advantages of MFP is that it was easy to build and test new pivoting schemes by using the same modularized subprograms for routine operations. Therefore, the relative merits of particular Gaussian Elimination pivot strategies could be easily evaluated. Also, factors such as fill-in and number of deletions could be used to check the effective use of data storage available. Since SIMULT similarly offered fill-in and deletion monitoring, the Gauss-Jordan algorithm by Key was included in all tests of the MFP variations.

Therefore, the objectives of the next phase were to compare SIMULT and MFP with more arbitrary matrices to find the "performance crossover point" (as suggested by the Mathematical Theory) and to find the optimum version of MFP which not only performs well but meets a prospective user's needs.

IV. The Choice of the Optimum Pivot Strategy

The initial test phase confirmed the programming logic for the three major Gaussian algorithms using standard test matrices; the next phase used matrices which were arbitrary both in value and structure. The range of structures included some systems which are typical problems in physics and engineering. Thus, the tests results and the choice of an optimum pivot strategy for Gaussian Elimination come as a result of the solutions of practical problems.

The discussion of the intermediate test phase includes a description of the test matrices, an analysis of the results with respect to the mathematical theory, and the process by which the final version of the MFP program (Gaussian Elimination) was developed.

Intermediate Test Matrices

All of the next eight test matrices started with randomly-generated numbers in a tridiagonal structure. About 35% of any particular set of numbers were negative. The last five matrices contained an additional 2% non-zero structure whose values were randomly generated; the coordinates for these extra values were also randomly determined. This additional structure was contained within bandwidths which normally ranged from ±5 to ±15 diagonals from the main diagonal. The last matrix, however, had its extra non-zero structure scattered throughout the entire available array.

These test matrices were used to exercise the variations of the Gaussian Elimination program (MFP) and the Gauss-Jordan Reduction program (SIMULT). The diagonal pivot strategy, MFP1, was not used in the intermediate phase; as the mathematical theory pointed out, diagonal pivoting strategy is really no strategy at all, and the chance exists that a zero would be found on a diagonal location. (The original purpose of MFP1 was merely to be the basic framework for the other pivoting strategies.)

The enumeration of the test matrices and their results with SIMULT and four variations of MFP are contained in Appendix E.

Analysis of the Results

The data which was available from the MFP strategies and the SIMULT program established three performance criteria: the order of magnitude of the error (as calculated according to Eq [29]), the number of times in which a fill-in value was manifested, and the execution time for the Gaussian algorithm.

Error Magnitudes. The most consistent performance was achieved by the Gaussian Partial Pivoting strategy (MFP2) with an error magnitude on the order of 10^{-12} or less (Table III). With Minimum Row/Minimum Column (MFP4), Minimum Row/Maximum Element (MFP5), and SIMULT, the error magnitudes were functions of the degree of "scatter" of the extra non-zeros: MFP4 and MFP5 (which were meant to reduce local fill-in) did work well with the more tightly banded matrices,

Table III
Error Magnitudes for Intermediate Tests

Test Matrix	SIMULT	MFP2	MFP3	MFP4	MFP5
# 5	10^{-2}	10^{-14}	10^{-1}	10^{-14}	10^{-14}
# 6	10^{-4}	10^{-14}	10^{-1}	10^{-14}	10^{-14}
# 7	10^{-1}	10^{-13}	10^{+1}	10^{-13}	10^{-13}
# 8	10^{-8}	10^{-12}	10^{0}	10^{-1}	10^{-1}
# 9	10^{-9}	10^{-12}	10^{0}	10^{0}	10^{0}
#10	10^{-8}	10^{-13}	10^{0}	10^{+4}	10^{+1}
#11	10^{-10}	10^{-13}	10^{0}	failed	10^{0}
#12	10^{-11}	10^{-13}	10^{+1}	10^{+1}	10^{0}

but did poorly with increasing disorder in the extra nonzero structure; SIMULT, on the other hand, clearly improved from 10^{-2} to 10^{-11} with more arbitrary structure. These observations clearly confirmed the predictions made in the Mathematical Theory. The failure of MFP4 with test matrix #11 was due to a computer diagnostic which stated that an "infinite operand" had been chosen. Since Minimum Row/Minimum Column pivots without regard to stability, this result is not surprising. As for Full Pivoting (MFP3), the error magnitudes were generally poor; this performance came chiefly as a result of excess fill-in.

Fill-in. With the exception of test matrix #12 (the least organised structure), Full Pivoting always created the most fill-in; furthermore, the fill-in excess was generally two or three times as much for the other Gaussian Elimination strategies (Table IV).

Table IV
Fill-in Tabulations for Intermediate Tests

Test Matrix	SIMULT	MFP2	MFP3	MFP4	MFP5
# 5	97	59	232	0	59
# 6	97	61	233	0	61
# 7	97	58	218	0	58
# 8	336	291	1149	207	249
# 9	987	699	2002	792	629
#10	871	930	2153	951	932
#11	1247	1044	2129	failed	717
#12	2168	4038	3815	2117	1713

For tightly-banded matrices, the strategies which pivoted for fill-in minimization did, in fact, fill in fewer values than the rest of the strategies; however, for more scattered structures, the reduction in local fill-in made little difference: that same local fill-in did come into play in many more calculations to manifest further fill-in; and, as with the "infinite operand" case, some values did become subsequent, unstable pivot elements.

There was a feature in the main program which would list the pivot ordering. The MFP4 and MFP5 programs often "jumped" around the matrix in successive pivots: row 1, then row 100; row 3, then row 89, for example. As a result, the fill-in lingered for many calculations and contributed to errors many more times than did the fill-in from the Partial Pivoting. For example, in the case of test matrix #12, in which Gaussian Partial Pivoting registered the greatest amount of fill-in, the new non-zeros were localized about the pivot elements, often eliminated soon after creation, and thus were not involved in as many subsequent calculations. This observation is in direct agreement with the Mathematical Theory.

Execution Time. The slowest of the MFP routines was always Full Pivoting (MFP3) because of the large number of extra computation required for the fill-in and the normally time-consuming searches for pivot elements. In the tightly-banded cases, Minimum Row/Minimum Column (MFP4) chose elements consecutively on the diagonal for pivoting, and thus computed very rapidly; similarly, Minimum Row/Maximum Element (MFP5) chose the same pivots as Partial Pivoting (MFP2). However, in general no Gaussian Elimination variation ran significantly faster than Partial Pivoting (Table V).

It is interesting to note that, while time comparisons between Gauss-Jordan and Gaussian Elimination are really not meaningful from an algorithmic standpoint, in the case of test matrix #12, only two orders of magnitude of accuracy separated SIMULT and MFP2; yet SIMULT solved the matrix nearly twelve times as fast as MFP2. Test matrix #12 is a very uncommon problem; but at some point, the potential user must decide which sparse solver he must choose in light of

Table V
Execution Times for Intermediate Tests

Test Matrix	SIMULT	MFP2	MFP3	MFP4	MFP5
# 5	0.26	0.36	0.73	0.37	0.44
# 6	0.27	0.39	0.76	0.37	0.45
# 7	0.30	0.36	0.74	0.43	0.45
# 8	0.33	1.02	3.40	1.00	1.03
# 9	0.66	1.82	7.77	2.38	1.90
#10	0.64	2.19	7.79	3.06	2.79
#11	0.89	2.46	9.17	failed	2.10
#12	1.70	21.11	26.82	9.55	6.35

Time in seconds.

the relative disarray of his own problem.

Choice of Optimum Algorithm

The choice of the Optimum Gaussian Elimination Algorithm was derived from the preceding analysis the conclusions of which are recapitulated below:

1. The best accuracy consistently came from Gaussian Partial Pivoting.

2. Disregard for stability in some problems led to poor accuracy and at least once case of division by a small number ("infinite operand").

3. By localizing all pivot choices (as in Partial Pivoting), the fill-in is also localized and its corresponding error affects many fewer subsequent calculations.

4. Large amounts of fill-in still may be an important source of error.

These conclusions suggested the following criteria for an "ideal" pivot strategy:

1. The pivot choice should be from consecutive rows; this choice would help to localize the effects of fill-in.

2. The column choice for that pivot row should initially attempt to minimize the number of calculations and, hence, lessen the probability for fill-in occurrence.

3. If, however, the value of the pivot is very small with respect to some number (called a "Pivot Tolerance") the element with the largest absolute value in the pivot row should be used as the pivot. This choice need only occur often enough to stabilize the system when instability insidiously appears.

As the dilemma of Fig. 5 indicates, even these ideals will not yield a panacea; however, they indeed provide adequate grounds for engineering tradeoffs among the criteria of accuracy, fill-in, and time.

Therefore, none of the original MFF variations was chosen as the optimum strategy; another strategy was developed, programmed and tested. This strategy was called "Consecutively Calculated" Pivoting. (The designation of this strategy is "MFPTH," and the subroutine name for the Forward Gaussian step is called "THINKER.")

The strategy of MFPTH is that suggested above: the row pivot coordinates go consecutively from 1 to n, and the column coordinate is chosen as that element in the column with the fewest number of non-zeros. However, if the value of the pivot is less than the pivot tolerance (called "PIVTOL") then a search is made to find the row element with the largest absolute value. The advantages of the strategy are very important to the user:

1. PIVTOL can be a readable quantity (as is the case in the listed program of MFPTH in the AFIT Computer Archive).

2. If the user is willing to sacrifice some accuracy, PIVTOL could be chosen to be a small number (0.01, for example) and the fill-in and number of calculations would be less than those for Gaussian Partial Pivoting.

3. On the other hand, if fill-in is not a problem, choice of a large value for PIVTOL (100, for example) would always be driving the system towards more stability.

In fact, with large values of PIVTOL, the algorithm, for all intents and purposes, is the same as Gaussian Partial Pivoting. There is, however, one important disadvantage. Use of a large PIVTOL would force more second searches for the pivot element; the user must therefore be willing to pay the price of extra time for extra accuracy.

The variation MFPTH was programmed and run; Table VI shows its performance with all test matrices as well as its core usage information.

The increase in time is apparent, but not formidable. It is clear that the MFPTH variation is a desirable program because the user has an input into the ultimate performance for his particular problem.

The next testing phase narrowed the scope of operation to problems likely to be solved in physics; the study also gave rise to yet another concept for the final form of the MFP Sparse Matrix Solver.

Table VI
Results of the Consecutively Calculated Strategy

Test Matrix	Error	Fill-ins	Deletions	Time (Seconds)
# 1	10^{-12}	0	297	1.08
# 2	10^{+1}	0	297	1.06
# 3	10^{-6}	0	297	1.07
# 4	10^{-12}	0	199	0.71
# 5	10^{-12}	0	199	0.73
# 6	10^{-14}	0	199	0.78
# 7	10^{-13}	0	199	0.75
# 8	10^{-9}	116	337	1.54
# 9	10^{-9}	455	474	2.84
#10	10^{-9}	620	570	3.58
#11	10^{-10}	715	602	4.08
#12	10^{-9}	2686	1103	26.56

PIVTOL = 0.01; THINKER Program length = 1202 (Octal)

V. The Final Algorithm Comparison

The final phase of testing compared the YSMP, SIMULT, and MFPTH sparse matrix solvers with eight more test matrices; the structures of these matrices were similar to the example presented in Chapter I: diagonally dominant, tridiagonal core structure with flanking diagonals. The analysis of these tests also spawned a new feature for the MFP program to improve speed. The discussion of the final phase, therefore, describes the new tests, presents the new features for MFP, tabulates the speed improvements, and sums up the overall performance of the three major sparse algorithms.

The Final Eight Test Matrices

The study of this final test phase concentrated on the performance of the sparse solvers with matrices whose flanking diagonals were originally located adjacent to the core (as in a pentadiagonal structure) and then displaced one diagonal at a time. (Appendix F contains a listing of these matrices.) The pivot tolerance for MFPTH was chosen to be 10 so as to pivot for accuracy. Table VII summarizes the error magnitudes and execution times for the three programs.

The obvious result of these tests is the consistent accuracy provided by the MFPTH program; however, because the pivot tolerance was large, the time needed to solve a system was relatively long for each test. What is not included in Table VII, but listed on the computer printouts, was that

Table VII
Final Performance Comparisons

Test Matrix		YSMP	SIMULT	MFPTH
#13	Error:	10^{-14}	10^{+45}	10^{-14}
	Time:	0.12	0.35	1.41
#14	Error:	10^{-2}	10^{+21}	10^{-14}
	Time:	0.16	0.34	2.13
#15	Error:	10^{-2}	10^{+12}	10^{-14}
	Time:	0.21	0.40	3.04
#16	Error:	10^{-2}	10^{0}	10^{-14}
	Time:	0.23	0.76	4.06
#17	Error:	10^{-2}	10^{-1}	10^{-14}
	Time:	0.27	1.05	4.96
#18	Error:	10^{-2}	10^{0}	10^{-14}
	Time:	0.28	0.78	6.09
#19	Error:	10^{-2}	10^{0}	10^{-14}
	Time:	0.30	1.11	7.03
#20	Error:	10^{-2}	10^{-5}	10^{-14}
	Time:	0.30	1.18	8.23

Notes: YSMP - LU Decomposition, a priori strategy.
SIMULT - Gauss-Jordan Reduction with Minimum Row/Minimum Column pivoting.
MFPTH - Gaussian Elimination with Consecutively Calculated pivoting (pivot tolerance = 10).

Error magnitudes calculated as in Eq (29).
Time in seconds.
ZTEST for SIMULT = 10^{-10}.

for each of the test matrices, the MFPTH pivoting strategy chose consecutive elements on the diagonals for the best accuracy. This observation suggested the next configuration of the MFP program.

Extra Features for MFP

Motivated by the diagonal pivot selections in the preceding tests and a desire to increase the speed of the MFPTH program, this student programmed an additional strategy. The new scheme was an a priori pivoting strategy which chose the pivot coordinates from the first non-zeros in consecutive rows; unless a zero appeared on the diagonal, then the entire diagonal was the source for pivot values. As an aid to the user, the a priori pivot subroutine (called "APRIORI") was included into the MFP program structure with the THINKER subroutine; as a result, the user was given an option for which strategy he desired. If a diagonally dominant system were being solved, choice of APRIORI would yield good accuracy with a relatively quicker solution time; if the APRIORI subroutine failed to give accuracy better than $R = 10^{-2}$ (Eq [29]), the main program would automatically reset and begin again with THINKER. Of course, THINKER could have been chosen from the beginning.

Speed Improvements in MFP

The solutions for each $\underline{x}$ using the a priori strategy gave precisely the same accuracy as in Table VII but with a measurable time improvement (Table VIII).

The designation for this configuration of MFP is "MFPOP" to indicate the "option" feature. (A listing of MFPOP can be obtained from the AFIT Computer Archive.) The additional core space required for APRIORI was only 370 (octal) locations; in terms of the entire program load size, the increase

Table VIII
Time Comparisons of APRIORI and THINKER

Test	Time:		
Matrix	APRIORI	THINKER	%Reduction
#13	1.18	1.41	16%
#14	2.11	2.13	1%
#15	3.02	3.04	1%
#16	3.99	4.06	2%
#17	4.90	4.96	1%
#18	5.83	6.09	4%
#19	6.78	7.03	4%
#20	7.87	8.23	4%

is negligible because many of the same FORTRAN system routines used by APRIORI were already present for THINKER.

Testing Conclusions

A user might be motivated to use the SIMULT program for dispersed, unstructured matrices or the YSMP program for tightly-banded matrices; this motivation comes as a result of time considerations. However, the MFPOP program demonstrated the capability to solve a very wide variety of matrices with consistently better accuracy than either SIMULT or YSMP. Also, the user's flexibility in controlling the progress of the solution with MFPOP is a very important consideration.

It must be emphasised that to attain the high degree of accuracy, the MFPOP program had to allow larger amounts of fill-in in both the a priori and local strategies than most classical sparse matrix studies conclude would be tolerable for a very large problem (n = 10,000, for example). The average speed improvement due to the introduction of the a priori pivoter was only 4%; but for a class of problems that are tightly banded, such as a pentadiagonal, the net improvement was a high 16%. With these factors of fill-in and time in mind, the next segment of this thesis addressed the problems of a new data-packing scheme and an even further improvement in the time factors.

VI. Sparse Packing in the CDC 6600 Computer

The choice of the MFPOP Sparse Solver as the optimum Gaussian Elimination algorithm came not only because of the higher theoretical accuracy which it provides but also because it was highly suitable for the CDC 6600 Computer. While all of the tested programs were written in the FORTRAN computer language (which is standard for most large computers), their accuracies were enhanced to a great degree by the numeric superiority which the CDC 6600 has over many computers. This chapter deals more closely with such computer capabilities as they pertain to a new packing scheme developed specifically for MFPOP; a consequence of this packing scheme is that it justifies the allowance for fill-in which, in many other computers, would be intolerable.

To help in the understanding of the new packing scheme, a review of the basic MFP packing method is presented, followed by the description of the implementation of the new packer. The final configuration of the Gaussian Elimination algorithm is also described since it is a streamlined version of the modular MFP concept. A summary presents the overall benefits of this student's program as it has been run on the CDC 6600 Computer.

Review of the Basic Packing Scheme

In order to pack only the non-zeros of the sparse A matrix, the MFP program needed several FORTRAN arrays. (The reader is directed to Appendix B for the complete packing method and the flow chart for the algorithm.) The arrays were as follows:

IA - The array (size N) which contains the locations of the starting points of the rows.

JA - The array which holds the column coordinates for each non-zero in the A matrix. (Size = the number of non-zeros, denoted "NNZR.")

A - The array which contains the non-zeros (size NNZR).

ISTAT - The array (size N) which contains the number of non-zeros in each row.

As part of the forward Gaussian algorithm, the arrays JSTAT (a column status vector), JCOL (a working vector), IPIV (row ordering array), and JPIV (column ordering array) were also required. Thus the minimum data space required for the sparse matrix A and its solution was the following set of arrays:

6 integer arrays, size N
1 integer array, size NNZR
1 floating-point array, size NNZR.

The size of NNZR for any routine must be judiciously chosen; a certain allowance for fill-in is required. As a rule of thumb, the following formula for NNZR was used in developing the final strategy:

$$\text{NNZR} \leq (5\%) \times (N^2) \times 2 \qquad (39)$$

To implement MFPOP on any computer, the information from each of the arrays is necessary; but one should note that if the largest number of equations to be solved is limited, the largest value stored in any of the elements of the integer arrays will be very small relative to the largest calculable integer for that computer. Since the CDC 6600 computer had a word size of 60 bits (which is nearly twice as large as the single precision word on most other computers), for an N on the order of 1000, only the right most ten bits of each word would be used; the remaining 50 bits would be wasted. It is the crux of the new packing scheme, therefore, to use as much of the integer word as possible to store information.

New Packing Scheme

The new packing scheme involves manipulation of the bits of the arrays for both the six N-sized arrays and the two NNZR-sized arrays.

N-sized Arrays. One can envision using the extra 50 bits of a computer word in the CDC 6600 as room to store other arrays; that is, by subdividing or "segmenting" the bit structure of the words in only one array, the information of many arrays can be compacted. To successfully implement this idea, the programmer must keep in mind that Octal arithmetic is used in the CDC 6600 and the storage and retrieval of information from a segmented word must be handled carefully.

In the modified MFPOP program, the six N-sized arrays are packed in groups of three: JSTAT, ISTAT, and IA; and JCOL, IPIV, and JPIV. The array variable name is called "INTBG" to indicate the integer data structure. The first

N values of INTEG contain JSTAT, ISTAT, and IA in three groups of 20 bits. The value of INTEG(N+1) contains a special value used by the packing subroutine, and the value of INTEG(N+2) holds the initial NNZR value. The next N locations in INTEG contain JCOL, IPIV, and JPIV in three groups of 20 bits.

Since the value of IA is usually a large number, the position of IA in the INTEG word is very important: since it occupies the right-most 20 bits, then the value of IA can be stored as if it were a decimal number. But such is not the case for JSTAT and ISTAT. However, these two status vectors are built, incremented, and decremented only one unit at a time; to add or subtract a "1" in the segment for ISTAT, a specific octal number (4000000_8) is added to the entire INTEG word. To continue the example, where the old statement was programmed as "ISTAT(I)=ISTAT(I) + 1," the new statement reads "INTEG(I)=INTEG(I) + 4000000B." (The "B" is the FORTRAN definition of an OCTAL constant.) A similar octal number increments JSTAT.

To extract the specific data for a given row K, the value of INTEG(K) is first placed into a working register. Then, by use of two supplied CDC functions, the proper information can be unambiguously retrieved: to extract IA(K), the value of INTEG(K) is masked with an ".AND." function over the right-most 20 bits; to extract ISTAT(K), the value of INTEG(K) is SHIFT-ed 20 bits to the right and then masked with the .AND. function (Ref 3:2-12; 8-4). For the arrays

JCOL, IPIV, and JPIV, a similar extraction method is used; however, to store the values, an intermediate register must come into play. The register is set to zero; the computed value of JCOL or IPIV is inserted and left-shifted the appropriate number of bits; then the contents of the register is added to the appropriate INTEG element. (Appendix G is the listing of the program with the bit-sliced packing.)

The savings on the N-sized arrays is very important for large linear systems; if N=1000, then the previously used 6000 locations for the arrays is reduced to only 2000. Thus the allowance of 4000 extra data locations is made available for fill-in. In practice, only 19 of the 20 bits per array are used; this restriction is necessary because the leftmost bits of JSTAT and JCOL are sign bits of the CDC 6600 computer words. Manipulation of the sign bit may create problems for the data stored in the entire word. With this configuration, the maximum number of equations is initially limited to $(2^{19}-1)$ or 524,287; as large as this number is, the NNZR-sized arrays place a much more stringent restriction on the maximum number of equations.

<u>NNZR-sized Arrays</u>. Further reduction in data storage requirements is also possible by combining the two NNZR-sized arrays, A and JA, since every A value has a corresponding JA table entry. The procedure is similar to the technique used in the N-sized arrays, except that floating-point numbers and integer numbers are mixed. In the CDC 6600 Computer, a floating-point number is stored with the

left-most 12 bits as the sign and the exponent, and the remaining 48 bits as the mantissa. In the new packing scheme, the right-most ten bits of the mantissa are masked to zero, and then the integer value of JA is inserted by using a logical ".OR." function. The array which takes the place of A and JA is called REALS.

The first N locations of REALS contain the b vector elements, and the next NNZR contain the compacted A and JA data. To retrieve a value for A, the value of REALS is fetched, and then the last ten bits are masked off; a real, floating-point number is the result. To retrieve a value of JA, the value of REALS has the left-most 50 bits masked, and an integer value is the result.

There is one very important advantage to this packing-feature: the storage required for each non-zero and each new fill-in is half of what the old scheme required. There are, however, three noteworthy disadvantages: 1) with only ten bits allowed for JA, then the maximum number of equations which can be solved is further restricted to (2^{10}-1) or 1023; 2) masking off ten bits from the floating-point number a_{ij} decreases the allowable accuracy to a maximum of only 11 decimal places instead of 14; 3) the real value stored in the mantissa is no longer rounded but truncated to 38 bits. While the maximum number of equations can be increased by changing the masks for A and JA (to 11, 12, or 13 bits) the maximum accuracy is accordingly decreased.

The sacrifice for accuracy, however, is not costly.

The MFP pivoting strategy can be very sensitive to stability, and thus it compensates for these induced machine inaccuracies. Table IX shows the performance changes manifested by the new packing strategy. (The new variation is called "GEBIT" to denote "Gaussian Elimination with Bit-slicing.")

Table IX
Comparison of MFPOP with GEBIT
on Some Test Matrices

Test	Errors:	
Matrix	MFPOP	GEBIT
# 1	10^{-12}	10^{-11}
# 3	10^{-6}	10^{-6}
# 4	10^{-12}	10^{-11}
# 6	10^{-14}	10^{-11}
#10	10^{-8}	10^{-10}
#15	10^{-14}	10^{-11}
#18	10^{-14}	10^{-11}
#20	10^{-14}	10^{-11}

Streamlined Algorithm

After the final testing of GEBIT configuration, an attempt was made to further improve the speed of the algorithm. As pointed out in Chapter III, the many calls to subroutines by the Gaussian Forward pivoters did use up much time; while this use of modular subroutines was a tremendous asset in the testing phase of this thesis, the final "production" model would have been unnecessarily slow. For the final configuration, therefore, the subroutines were removed, and their logic structures were programmed within the Gaussian subroutines THINKER, APRIORI, and GAUSSBX. This program modification also saved extra time in that only the logic necessary at a particular step in the algorithm was used and not the all-encompassing logic of the subroutines DELETE, STORE, and FETCH (Appendix B). The numerical accuracy of this final configuration is naturally the same for GEBIT; but Table X shows the improvement in the time usage over GEBIT and MFPOP. (The final configuration is called "SMART" to denote the "Sparse Matrix Algorithm Research Thesis.")

Table X
Time Improvement of the Streamlined Algorithm

Test Matrix	Time (Seconds): MFPOP	GEBIT	SMART
#13	1.18	1.04	0.75
#14	2.11	1.77	1.39
#15	3.02	2.50	2.02
#16	3.99	3.25	2.67
#17	4.91	4.03	3.10
#18	5.83	4.75	4.01
#19	6.78	5.57	4.69
#20	7.87	6.50	5.41

Interestingly enough, the GEBIT configuration showed an average improvement of nearly 17% by itself. One reason for this increase is that the time for the fast register functions (SHIFT, .AND., and .OR.) are much less than the fetch commands from the relatively slow core memory. Thus the computer needs only to fetch one number, REALS(M), to attain both the values for JA(M) and A(M). Overall, the SMART configuration showed an improvement of nearly 33% in time.

Production Model Summary

While the SMART routine was still slower than Sherman's YSMP or Key's SIMULT, the accuracy demonstrated that this program can be a competitive sparse matrix solver for very large programs. A scaled version of SMART was run on the AFIT CDC 6600 computer with a 1000-by-1000 pentadiagonal matrix. The error was on the order of 10^{-10}; but the most obvious result was the total core usage required: only 64K words. With the packing schemes of YSMP and SIMULT, to run the same problem would have required considerably more core storage (well over 100K words). For problems not quite as large as N=1000, the SMART program could be used on the INTERCOM terminals at AFIT and AFWL where the core limitation is set at 60K. Clearly, the Gaussian Elimination programmed for this thesis is an adequate computational tool.

VII. Conclusions and Recommendations

The summary of this thesis includes a discussion of the attainment of the thesis objectives, a discussion of the outcome of the algorithm programmed by this student, and a list of subject areas for further study.

Attainment of Thesis Objectives

The comparison of the two existing sparse matrix solvers (the LU Decomposition in YSMP and the Gauss-Jordan Reduction in SIMULT) pointed out two clearly definable areas where the accuracy of one program was much better than that of the other: YSMP worked well for tightly-banded matrices while SIMULT favored the more unstructured systems. Unfortunately, the areas in which the programs worked their best did not overlap; furthermore, the matrix structure of the example in Chapter I--a very common structure--was not contained in either region.

The correlative program of the Gaussian Elimination with strategic pivoting (called SMART) not only filled the void but included areas of performance common to both YSMP and SIMULT. The pivot schemes available with SMART (Consecutively Calculated or an a priori pivot) yield a potentially serious consequence: the possibility exists that fill-in may occur more frequently than in the programs of other methods. In fact, these pivot strategies conflict somewhat with the thoughts of the writers of many articles in Sparse Matrix literature (i.e., there is a great emphasis on fill-in minimization in many of these papers). This apparent conflict was resolved in two ways which justify the use of these pivot schemes: 1) by an examination of the word structure in the computer at hand and 2) by the construction of the new packing scheme for this thesis.

Computer Word Structure. The preponderance of computers used for sparse matrix study in the past decade have only 32-bit, single-precision words; to attain good accuracy for large sparse systems, the use of double-precision FORTRAN is a necessity. Therefore, the total number of distinct storage locations is cut in half. Thus the concern for fill-in growth in these kinds of computers is quite valid. On the other hand, the CDC 6600 provides a 60-bit, single-precision word which allows essentially the same accuracy as most 32-bit-per-word computers at double-precision. As a result, much more extra space is available if fill-in grows to large proportions.

Packing Scheme. Additionally, the bit-sliced packing scheme used in SMART further reduces the requirements for data storage by combining information from several data arrays into a single data array. In handling the compacted $\underline{A}$ matrix alone, the SMART algorithm uses nearly 50% less storage space by combining the $\underline{A}$ values and their respective column index pointers into divisions of the same arrays. (This new packing scheme thus fulfils the second major objective of this thesis: the efficient use of computer core storage.)

Thus any conflict with the proponents of fill-in reduction is avoided because the SMART program as it is run on the CDC 6600 computer can clearly be allowed to pivot for accuracy or calculation reduction rather than strictly for fill-in minimization: the extra core space is readily available.

Critique of the Gaussian Elimination Program

There are several factors to discuss about the Gaussian Elimination algorithm as it is programmed in SMART; the factors discussed deal with both tangible and intangible considerations.

Tangible Advantages. The accuracy of SMART is well-documented over a wide scope of linear systems. Even for very large systems, the core usage for SMART is small when compared to the two existing sparse solvers tested. Also, inasmuch as the user has an option on the pivot strategy (consecutively calculated or a priori) and a choice for pivot tolerance (for accuracy or some fill-in reduction), the SMART program can satisfy a large range of required capabilities very easily.

Tangible Disadvantage. The single drawback of SMART is the time it takes to solve a problem. The reason for this time excess is related to the packing and core usage designs: to keep core usage low, much of the array of data must be relocated at each incidence of a fill-in or deletion. For example, if a fill-in were to occur near the top of the data array, the subsequent data are all shifted down one location to make room for the fill-in. The array is similarly shifted up one location for each deletion. In the context of a large linear system whose solution may require some fill-in, these data manipulations for the sake of core storage translate into much extra computer execution time.

Intangible Advantages. Even with long execution times, programs which require less core space in a computer are often given higher priority for execution; as a result, the output from the SMART program would be finished and into the hands of the user much sooner than for YSMP or SIMULT for the same large, sparse matrix. This "turn around time" can be a very important element in a user's computational needs. (The proper analysis of this concept lies in understanding the operating system of the particular computer in use.) In any case, in a large multiprogrammed computer environment (such as AFIT or AFWL), it is generally harder to get large amounts of core at a given time than it is to get extended execution time. Another intangible benefit of SMART is that Gaussian Elimination is very easily studied; thus the algorithm makes SMART highly adaptable to other thesis study.

Suggested Areas for Further Study

In the entire field of Sparse Matrix research, there are other considerations which have been applied to other sparse matrix solution techniques. As a way to mention some of these factors and how they might apply to the SMART algorithm, the following recommendations are presented:

Reduction of Execution Time. 1) It would be interesting to attempt a Consecutively Calculated Pivoting strategy in the faster Gauss-Jordan Reduction and LU Decomposition programs of Key and Sherman. 2) The packing scheme of SMART could be modified so that the time-consuming "shuffle" during deletions and fill-in could be eliminated: by use of a "linked list" table, elements of a row need not be stored adjacent to each other; the table list would contain the computer "addresses" of consecutive row elements. Bit-slicing could be applied here so that the linked list table would not require inordinate amounts of extra working space.

Iterative Techniques. 1) While this thesis dealt only with direct solutions of sparse matrices, it would be appropriate to conduct a thorough investigation of iterative techniques and compare their results with SMART. 2) Many writers suggest using iterative improvers for systems which are solved with poor accuracy: the direct solutions can be used as the starting points for the iterations. Along these lines, consideration should be given to the case where a particular solution to Eq (1) is found the first time using SMART; then, some elements of $\underset{\sim}{A}$ or $\underset{\sim}{b}$ might be changed to reflect subtle differences in the mathematical model. An iterative solver could use the first accurate solution from SMART as a starting point for the solution of the modified system. Such a technique will provide much faster solutions than complete recomputation by either SMART or the iterative solver starting from scratch.

Rigorous Mathematical Tests. 1) For a class of near-singular matrix systems, it may be necessary to interface a direct solution with an iterative solution as described above; the useful result would be an idea of the true extent for which the Gaussian sparse solutions are applicable.
2) In order to handle problems which transcend the well-conditioned systems common to physics and engineering, the programming of a scaling algorithm (as in Eq [34]) would allow the more theoretical systems to be tested. 3) Additional study on algorithms for special matrix structures (such as symmetic or symmetric zero structure) is a naturally follow-on to general matrix solutions.

Program Adaptability. 1) Since sparse matrices can come as a result of finite differencing techniques for solving differential equations, it may be appropriate to construct computer programs which can generate the sparse matrices given the differential equations and the boundary conditions. An important suggestion in this case is to standardize the data formats so that the sparse generator programs can interface directly with the sparse solver.
2) To carry the analogy one step further, one can conceive of one large computer program which generates the sparse matrix, solves it with direct methods, and improves the answer with iterative techniques. With such a large concept, exploitation of the computer's operation system would be a valuable aid to this end.

Use of the SMART Algorithm on Other Computers. It would be of great value to attempt to execute SMART both in single-precision and double-precision on a computer other than the CDC 6600; the performance degradation by using a 32-bit-per-word computer would be of particular interest. It would be necessary to make changes to the bit-sliced array packing depending on the sophistication of a particular FORTRAN compiler with such computers.

Concluding Statement

In summary, the most important result of this thesis is the development of the sparse matrix solver SMART. The program gives the user much flexibility in the conduct of a particular problem's solution. The algorithm of SMART programmed on the CDC 6600 computer can provide accurate solutions from a very compact, efficiently used core structure for a wide range of linear system structures.

Bibliography

1. Brandon, D. M., Jr. "The Implementation and Use of Sparse Matrix Techniques in General Simulation Programs." The Computer Journal, 17:165-171 (May 1974).

2. Bunch, J. R. "Analysis of Sparse Elimination." SIAM Journal on Numerical Analysis, 11:847-873 (October 1974).

3. Control Data Corporation, FORTRAN Extended Version 4 Reference Manual. Sunnyvale, California, 1977.

4. Dantzig, G. B., et al. Sparse Matrix Techniques in Two Mathematical Programming Codes. Proceedings of a Symposium on Sparse Matrices and Their Applications, held September 9-10, 1968, at the IBM Watson Research Center, Yorktown Heights, New York. New York: Plenum Press, 1969.

5. Datapro Research Corp. Datapro 70, the EDP Buyer's Guide. Vol. I. Delran, New Jersey: McGraw-Hill Co., 1976.

6. Duff, I. S. "A Survey of Sparse Matrix Research." Proceedings of the IEEE, 65:500-535 (April 1977).

7. Forsythe, G. E. and C. E. Moler. Computer Solutions of Linear Algebraic Systems. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1967.

8. Gustavson, F. G. Some Basic Techniques for Solving Sparse Systems of Linear Equations. Proceedings of a Symposium on Sparse Matrices and Their Applications, held September 9-10, 1971, at the IBM Watson Research Center, Yorktown Heights, New York. New York: Plenum Press, 1972.

9. Key, J. E. Computer Program for Solution of Large, Sparse, Unsymmetric Systems of Linear Equations. AFOSR-TR-F44620-69-C-0124, August 1972.

10. Murphy, H. M., Jr. TIMER. (A Special computer function programmed for the CDC 6600 Computer at Kirtland AFB, N.M.) December 1973.

11. Myers, G. E. Analytical Methods in Conduction Heat Transfer. New York: McGraw-Hill, Inc., 1971.

12. Peters, G. and J. H. Wilkinson. "On the Stability of Gauss-Jordan Elimination with Pivoting." Communications of the ACM, 18:20-24 (January 1975).

13. Ralston, Anthony. A First Course in Numerical Analysis. New York: McGraw-Hill, Inc., 1965.

14. Sherman, A. H. Yale Sparse Matrix Package User's Guide. Lawrence Livermore Laboratory, August 1975. UCID-30114.

15. Tewarson, R. P. Sparse Matrices. New York: Academic Press, 1973.

APPENDIX A

Example of the Need for Strategic Pivoting

Appendix A

Example of the Need for Strategic Pivoting

Given: $0.0001\ x_1 + 1.00\ x_2 = 1.00$

$1.00\ x_1 + 1.00\ x_2 = 2.00$

Desk calculator solution: $x_1 = 1.00010001$

$x_2 = 0.99989998$

"Machine" limitation: Accuracy is limited
to three places.

Case I: Gaussian Elimination without Pivoting

Operation 1 - $1.00\ x_1 + 10{,}000\ x_2 = 10{,}000$

$0.0\ x_1 - 9{,}999\ x_2 = -9{,}998$

Operation 2 - $1.00\ x_1 + 10{,}000\ x_2 = 10{,}000$

$0.0\ x_1 + 1.00\ x_2 =$ "1.00" (round-off)

Solution: $x_1 = 0.00$ and $x_2 = 1.00$

Case II: Elimination with Row Interchange

$1.00\ x_1 + 1.00\ x_2 = 2.00$

$0.0001\ x_1 + 1.00\ x_2 = 1.00$

Operation 1 - $1.00\ x_1 + 1.00\ x_2 = 2.00$

$0.0\ x_1 +$ "1.00" $x_2 = 1.00$
(round-off)

Solution: $x_1 = 1.00$ and $x_2 = 1.00$ ("Perfect" to within
round-off accuracy
of desk calculator.)

(From Ref 7:34)

APPENDIX B

Flow Charts for MFP Subroutines

Appendix B

Flow Charts for MFP Subroutines

General Information

This appendix contains the logical flow charts for the important subroutines in the MFP sparse matrix solver: PACK1, FETCH, STORE, DELETE, GSSGEN, and GAUSSBX. The program GSSGEN is the forward Gaussian Elimination with a general pivot strategy which gave rise to the variations of MFP. (A FORTRAN listing of the entire MFP program can be obtained from the AFIT Computer Archive.)

PACK1

PACK1 compresses the A matrix into a compact form, reads in the b vector, and generates status information. The FORTRAN arrays used are the following:

IA - The starting address of the i-th row.
JA - The column coordinates for the A values.
ISTAT - The number of non-zeros in the i-th row.
A - The column array of the A matrix in compressed form.
B - The constant vector.

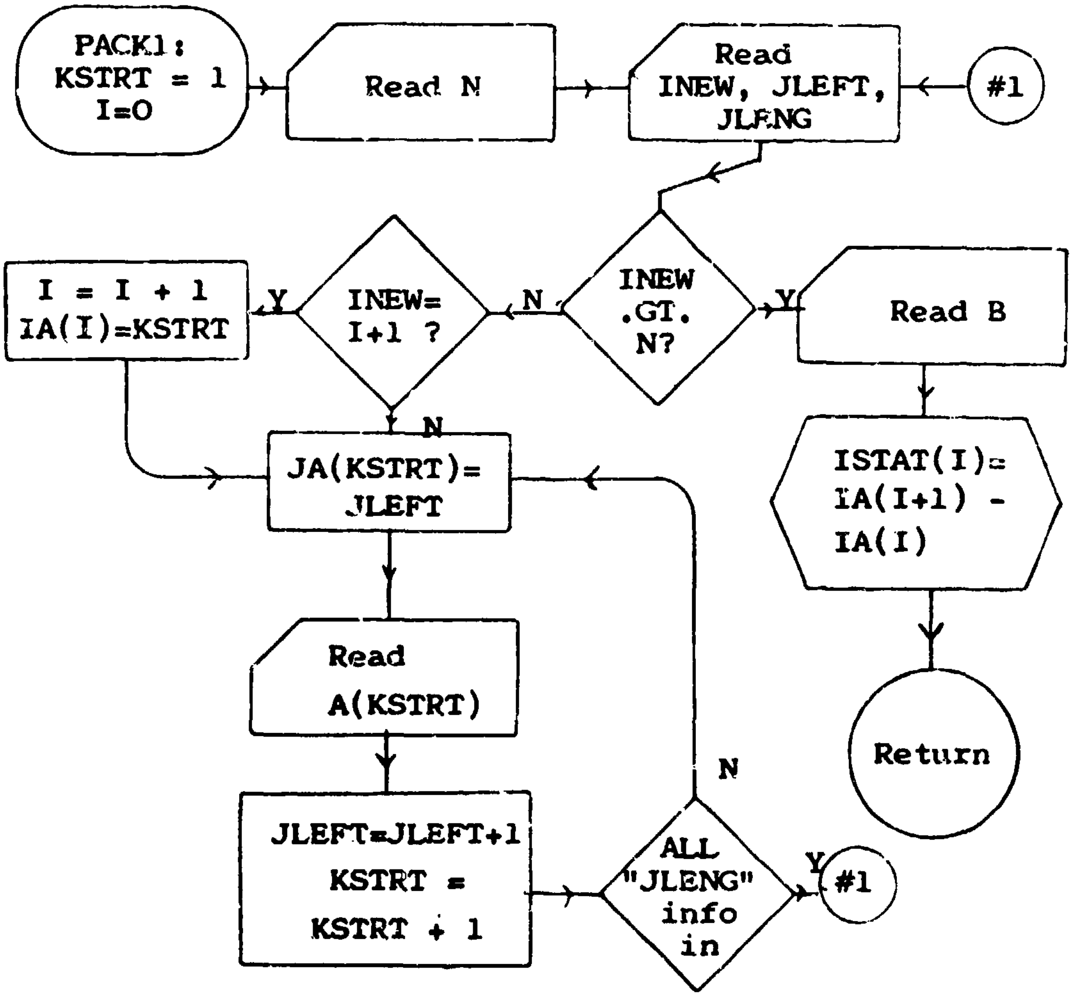
Fig. B-1 shows the packing of only the non-zeros in a sample 4-by-4 system. The flow chart (Fig. B-2) depicts only the essential logic for packing and not the error checks which are contained in the program.

The main program and all of the functional subroutines use the arrays generated in PACK1. This information is exchanged between programs by way of a COMMON statement.

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 \\ 0 & 0 & 7 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Index	ISTAT	IA	JA	A	B
1	2	1	1	1	1
2	2	3	2	2	2
3	2	5	2	3	3
4	2	7	3	4	4
5		9	2	5	
6			4	6	
7			3	7	
8			4	8	

Fig. B-1. Packing Scheme for MFP



INEW - The row number of the next set of data.
JLEFT - The leftmost coordinate of first piece of data.
JLENG - How many non-zeros will be read.
KSTRT - The staring address of the next row.
I - The index of the current row being packed.
IA(N+1) - The location after the last non-zero.

Fig. B-2. Flow Chart of PACK1

FETCH

To attain the value of an a_{ij}, a call to FETCH will return with the value or zero if a value is not found. Since a comparison with a floating point zero in FORTRAN is not always valid, a logic flag called ZERO is set to TRUE if no value is found; this actually saves time since a logic check is faster than an arithmetic comparison. The call to subroutine requires the coordinates I and J; the value, the condition of the logic flag, and a value called IHOLD are returned. In subsequent subroutines, IHOLD is used because it contains the present address of a_{ij}; if a_{ij} must subsequently be deleted or a new value stored into a_{ij}, IHOLD tells immediately where that value must go. Thus, the program is spared the extra file search for the location of a_{ij} for these other subroutines.

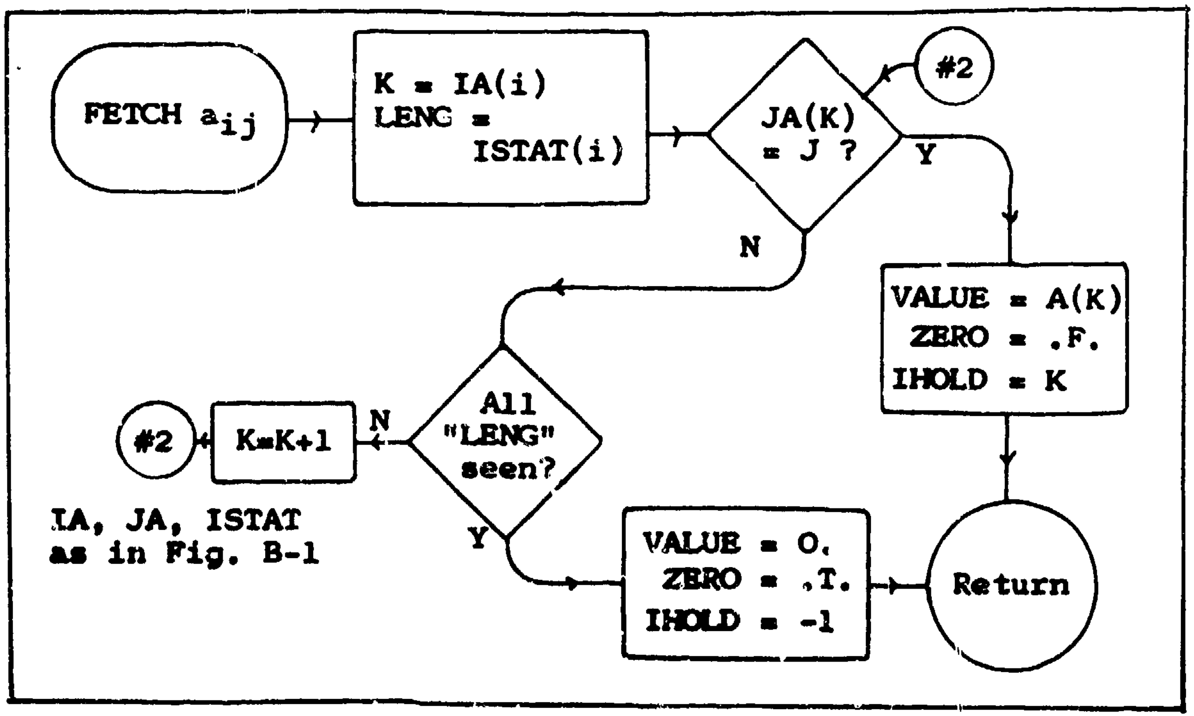


Fig. B-3. Flow Chart of FETCH

DELETE

The subroutine DELETE removes from the compacted form of the A matrix any value which will be eliminateu in the Gaussian forward process or any pivot element which has been normalized and is understood to be exactly 1.0 in value.

Any element a_{ij} can be eliminated by DELETE; however, it is usually the case that the value IHOLD contains a number which is the address of the a_{ij} to be eliminated. Thus another search of the row is not necessary. There is a safety check: the coordinate J is checked with the value of JA(IHOLD) to be sure that the correct element is to be eliminated. If this test fails, the subroutine merely reverts to a row search. The following variables are defined for use in the flow chart (Fig. B-4):

NDEL - The address of the value to be deleted.

IDEL - A counter to track the number of deletions which the forward Gauss subroutine must make.

Once NDEL is computed, the arrays JA and A are all shuffled up one location starting at JA(NDEL) and A(NDEL). Then the starting addresses of rows (i+1) through (n+1) are decremented one place.

The counter IDEL can be used to check the relative performance of the various pivoting strategies. Once the final form of MFP has been established, the counting of IDEL is no longer needed for the user's information.

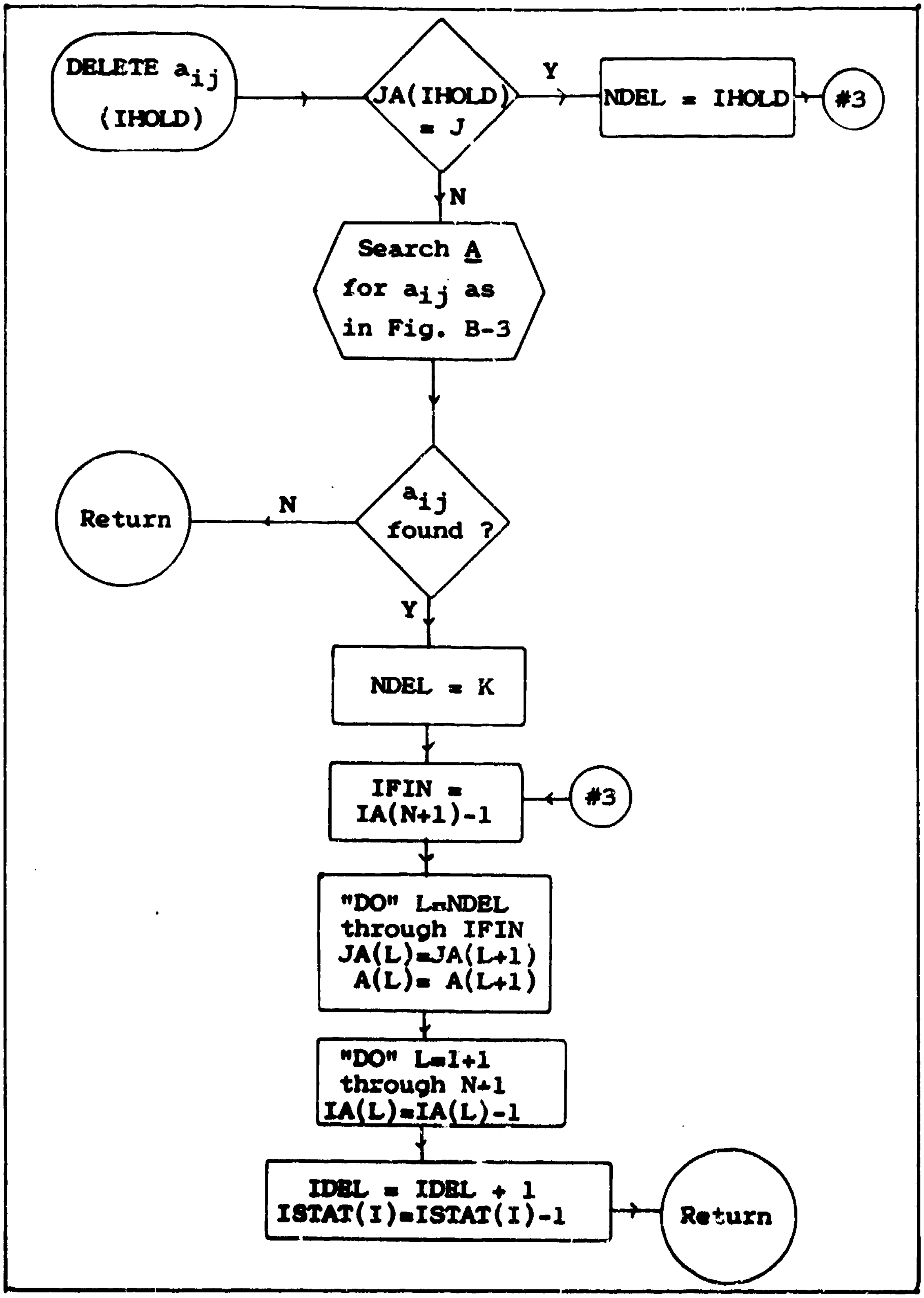

DELETE a_{ij} (IHOLD)
JA(IHOLD) = J
Y
NDEL = IHOLD
#3
N
Search A for a_{ij} as in Fig. B-3
Return
N
a_{ij} found ?
Y
NDEL = K
IFIN = IA(N+1)-1
#3
"DO" L=NDEL through IFIN JA(L)=JA(L+1) A(L)= A(L+1)
"DO" L=I+1 through N+1 IA(L)=IA(L)-1
IDEL = IDEL + 1 ISTAT(I)=ISTAT(I)-1
Return

Figure B-4. Flow Chart for DELETE

STORE

The subroutine STORE places a computed value back into the A matrix or, if the situation dictates, it will create space for a fill-in value.

As with the subroutine DELETE, it is usually the case that the value of IHOLD contains the address of the a_{ij} where the data must be stored; a similar safety check is performed. If the safety check fails, a standard row search is completed. After a thorough search, the subroutine defaults into the "fill-in" mode.

For fill-in, the program must find the coordinate before which the data must be entered; then, all of the subsequent data are shuffled down one location. The new data value is inserted into the empty space in both the A and JA array tables. The variables which are used in the flow chart (Fig. B-5) are the same standard set plus the following:

IBUMP - The coordinate at which the fill-in will be placed.

IFILL - A counter to track the number of fill-in values which occur in the Forward Gaussian process.

The starting address vector, IA, is finally incremented by one for each row after the fill-in row.

The counter IFILL can be used to check the relative performance of the various pivoting strategies; the counter is no longer needed for the final version of the program.

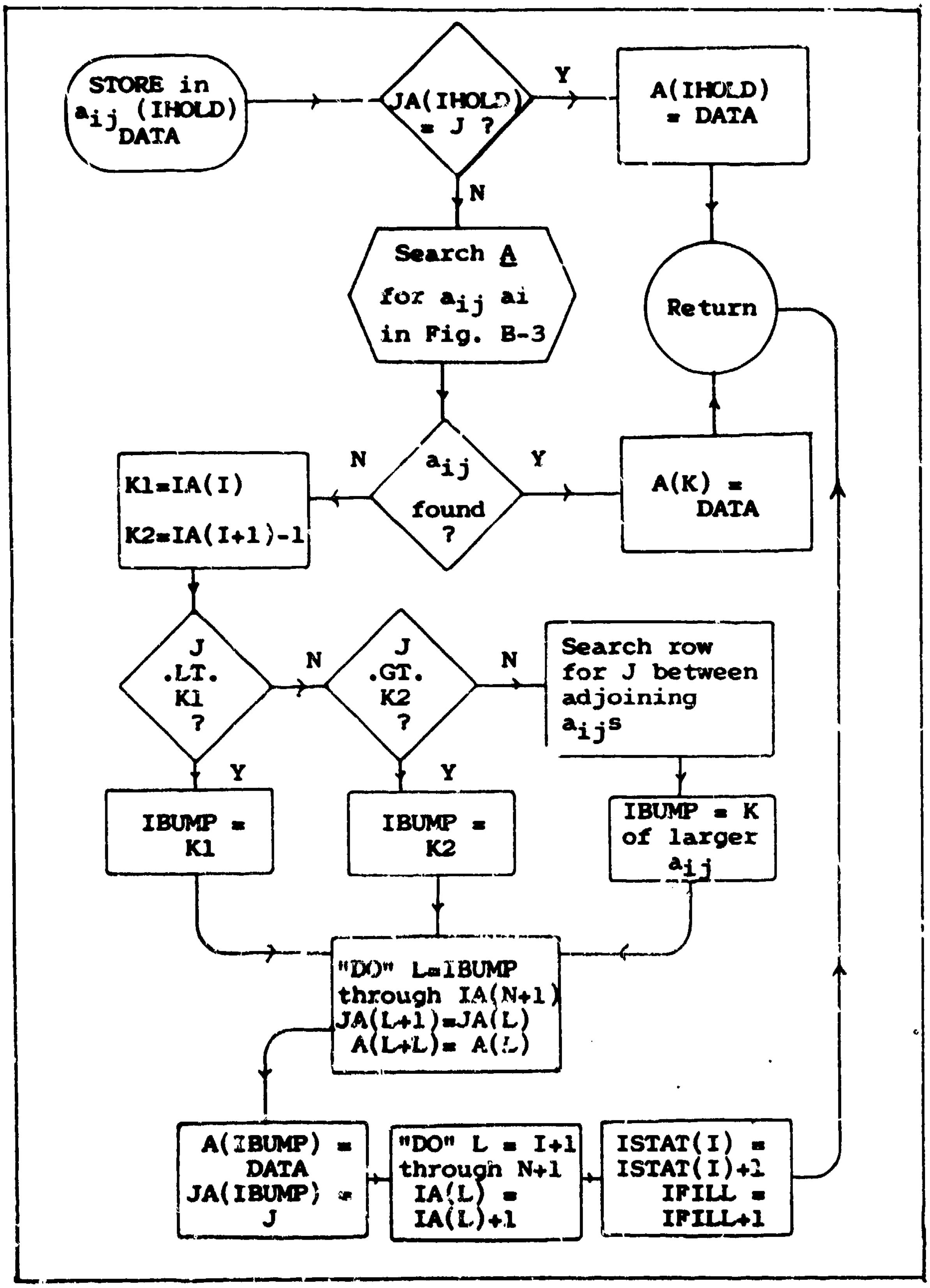

STORE in a_{ij} (IHOLD) DATA
JA(IHOLD) = J ?
Y
A(IHOLD) = DATA
N
Search A for a_{ij} ai in Fig. B-3
Return
a_{ij} found ?
N
Y
K1=IA(I)
K2=IA(I+1)-1
A(K) = DATA
J .LT. K1 ?
N
J .GT. K2 ?
N
Search row for J between adjoining a_{ij}s
Y
Y
IBUMP = K1
IBUMP = K2
IBUMP = K of larger a_{ij}
"DO" L=IBUMP through IA(N+1) JA(L+1)=JA(L) A(L+L)= A(L)
A(IBUMP) = DATA JA(IBUMP) = J
"DO" L = I+1 through N+1 IA(L) = IA(L)+1
ISTAT(I) = ISTAT(I)+1 IFILL = IFILL+1

Fig. B-5. Flow Chart of STORE

GSSGEN

The subroutine GSSGEN accomplishes the forward Gaussian Elimination step. This subroutine is really an illustrative example in that the flow chart (Figs. B-6 and B-7) indicates pivoting based on some arbitrary strategy. The following FORTRAN variables are defined:

ELIM(I) - An element of an array ELIM which is declared as a LOGICAL variable name. If ELIM(I) is .TRUE., then row I has been used as a pivot row and should not be used for substitution. If ELIM(I) is .FALSE., then row I is a candidate for a pivot or a substitution.

IELROW - The row number of the current pivot row.

JEL - The column number of the current pivot column.

IPIV - An array which orders the pivot rows for the back-solver.

JPIV - An array which orders the pivot columns for the back-solver.

JSTAT(I) - The number of non-zeros in the I-th column. This status vector is used by Minimum Row/Minimum Column pivot strategy, for example.

If the diagonal pivot strategy were used, there would be no need for JSTAT or ELIM; in this case, IELROW and JEL would always equal the value K. But in more complicated strategies, these variables are necessities.

IPIV and JPIV are used so that no row or column exchanges are necessary in a pivoting strategy. Their utility is described further in the description of GAUSSBX.

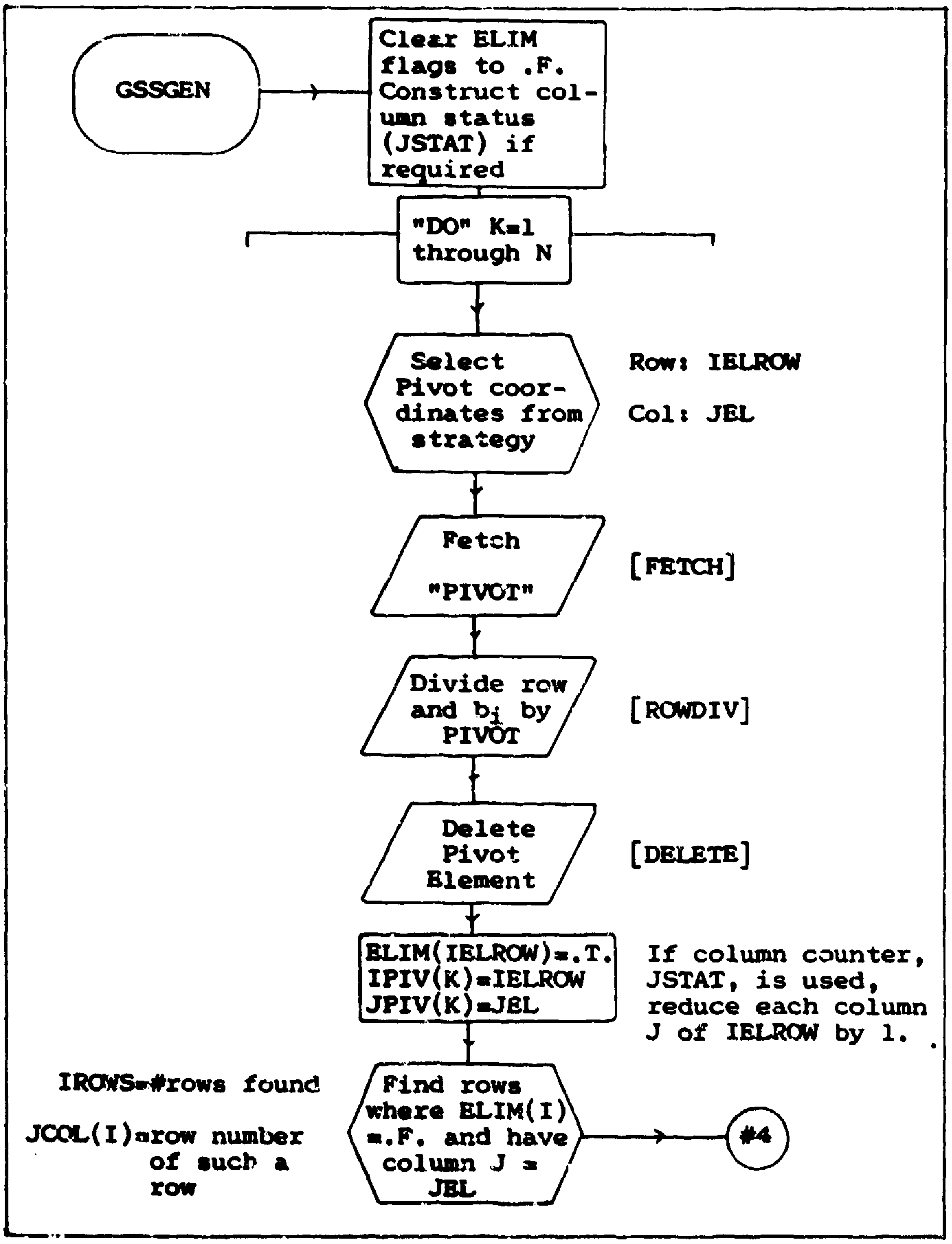

GSSGEN
Clear ELIM flags to .F. Construct column status (JSTAT) if required
"DO" K=1 through N
Select Pivot coordinates from strategy
Row: IELROW
Col: JEL
Fetch "PIVOT"
[FETCH]
Divide row and b_i by PIVOT
[ROWDIV]
Delete Pivot Element
[DELETE]
ELIM(IELROW)=.T.
IPIV(K)=IELROW
JPIV(K)=JEL
If column counter, JSTAT, is used, reduce each column J of IELROW by 1.
IROWS=#rows found
JCOL(I)=row number of such a row
Find rows where ELIM(I) =.F. and have column J = JEL
#4

Fig. B-6. Search Portion of GSSGEN

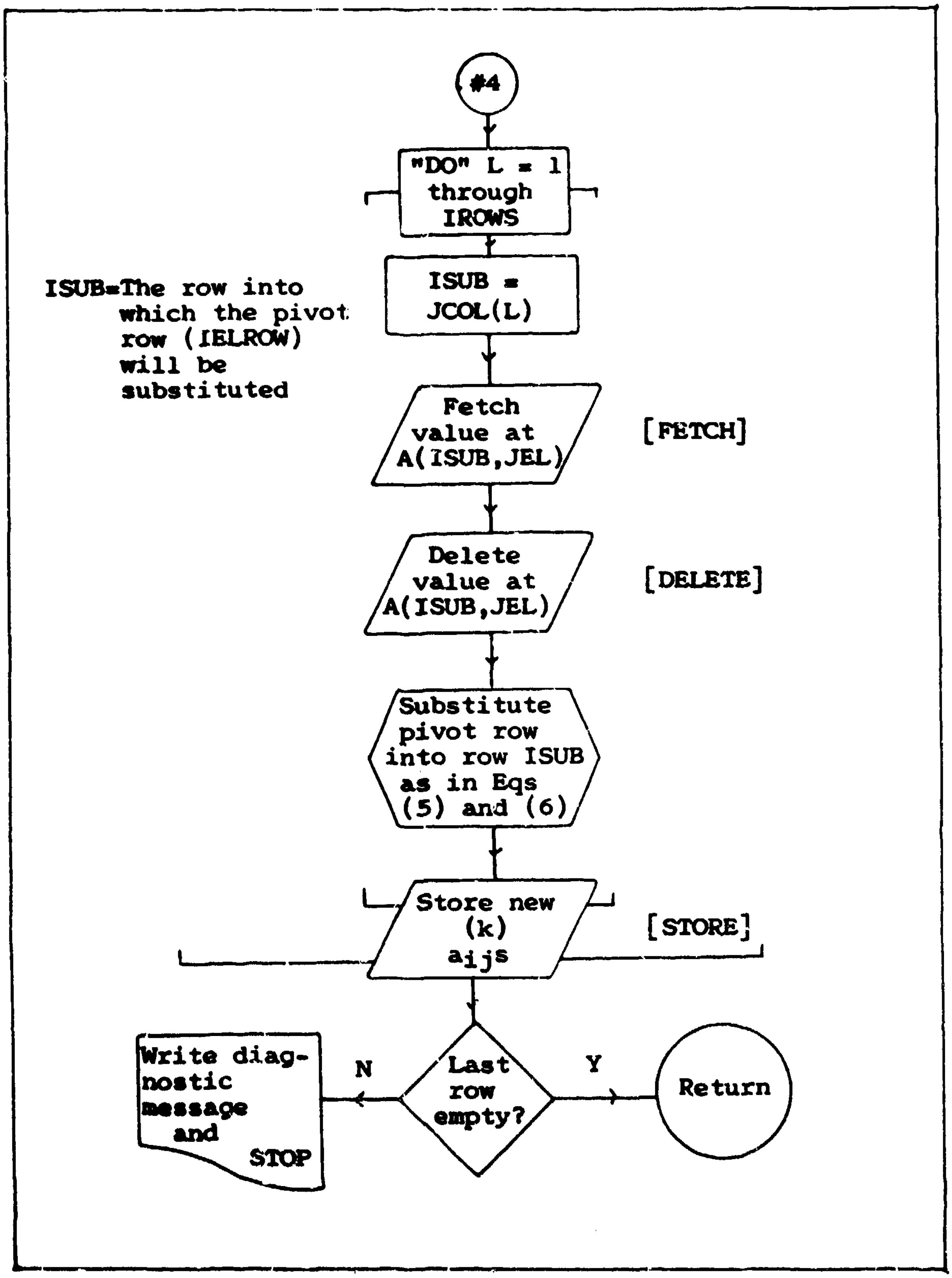

#4
"DO" L = 1 through IROWS
ISUB=The row into which the pivot row (IELROW) will be substituted
ISUB = JCOL(L)
Fetch value at A(ISUB,JEL)
[FETCH]
Delete value at A(ISUB,JEL)
[DELETE]
Substitute pivot row into row ISUB as in Eqs (5) and (6)
Store new (k) aijs
[STORE]
Write diagnostic message and STOP
N
Last row empty?
Y
Return

Fig. B-7. Substitutional Portion of GSSGEN

GAUSSBX

The back solution of the Gaussian Elimination is done by the GAUSSBX subroutine. The operation corresponds to Eq (7). In the forward solution, the ordering of rows and columns is stored in the arrays IPIV and JPIV. The flow chart for GAUSSBX (Fig. B-8) shows how the proper x_i values are computed. In the example of Fig. B-1, if the second row, third column were the last pivot coordinates, then IPIV(N) and JPIV(N) would be "2" and "3" respectively. The first x_i to be solved would therefore be x_3. Had the rows and columns been interchanged, x_3 would have appeared in the last element in the $\underline{x}$ vector. Thus, the use of row and column pivot arrays saves much extra programming and execution time required by interchanges in $\underline{A}$ and $\underline{x}$.

Subroutine Summary

The calling sequence for a driver program for these subroutines would be as follows:

PACK1 - To store the compressed data.
GSSGEN - For the forward solution.
GAUSSBX - For the back solution.

There are other subroutines used in the MFP program; but their structure is very simple, and reference to the listing would be sufficient for further study.

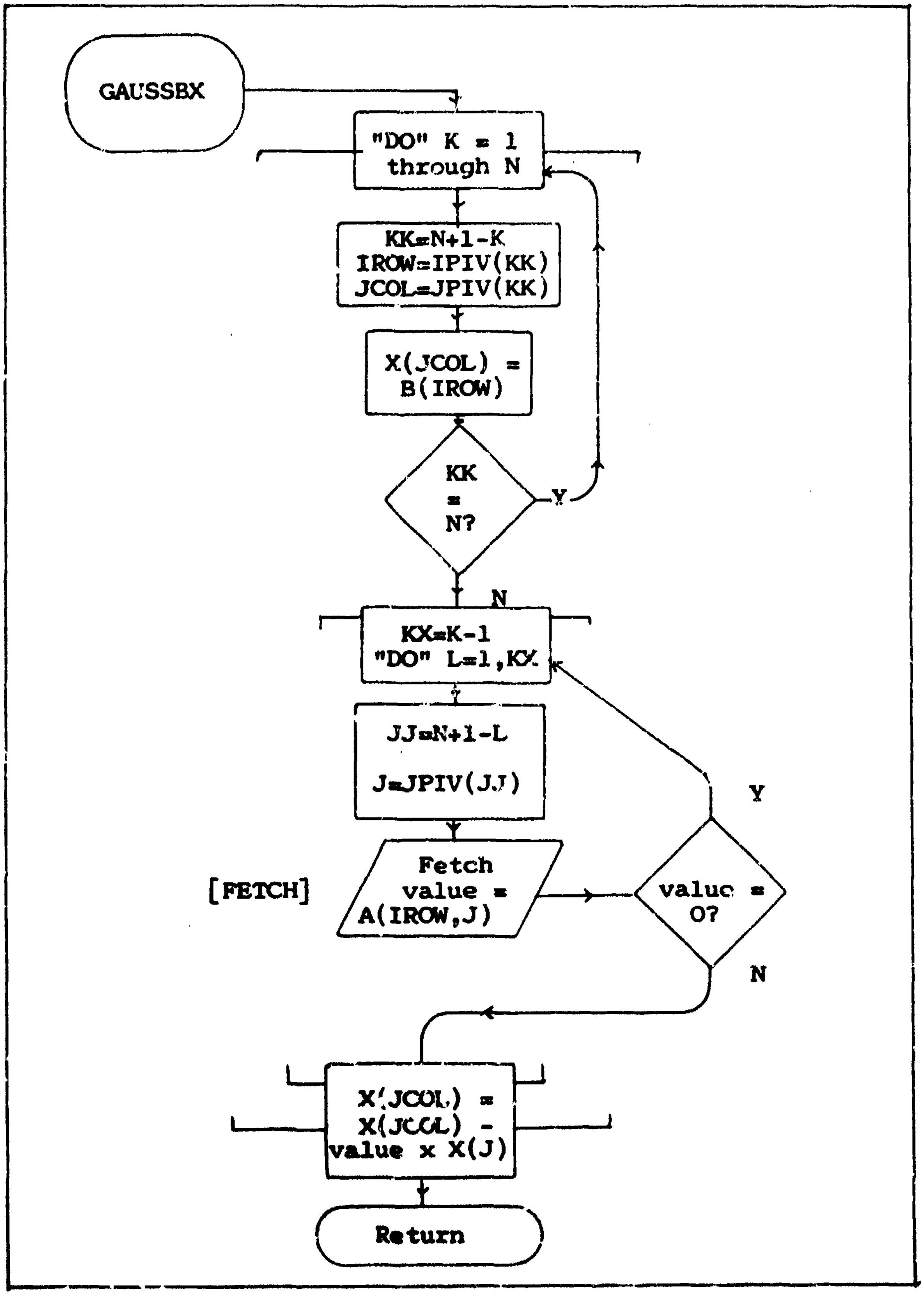


Fig. B-8. Flow Chart for GAUSSBX

APPENDIX C

Core Storage for Gaussian Sparse Solvers

Appendix C

Table C-I
Core Storage for Gaussian Sparse Solvers

Program	Subroutine	Length (Octal)	Data Length (Octal)
YSMP -	SORDER	411	
	NSRORD	151	
	SSFAC	351	
	NSNFAC	321	
	NSBSLV	141	
	ZEROSYM	317	
	Total:	2134	20207
SIMULT -	SIMULT	432	
	PIVSEL	100	
	Total:	532	17662
MFP -	FETCH	31	
	ROWDIV	22	
	DELETE	57	
	STORE	134	
	GAUSSBX	56	
#1	GAUSSF - Diagonal	413	
	Total:	757	16666
#2	GAUSSFP - Partial	417	
	Total:	763	16666
#3	GAUSSFT - Full	570	
	Total:	1034	16666
#4	GAUSSMM - Min Row Min Col	771	
	Total	1335	16666
#5	GAUSSML - Min Row Max ele	601	
	Total:	1145	16666

Storage for any arbitrary 100-by-100 matrix, 5% sparsity.

APPENDIX D

Standard Test Matrices

Appendix D

Standard Test Matrices

The following matrices (Figs. D-1 through D-4) were used in the initial test phase of this thesis.

Specifications

In all cases, the rank, n, was 100, and all elements of the constant vector, b, were unity.

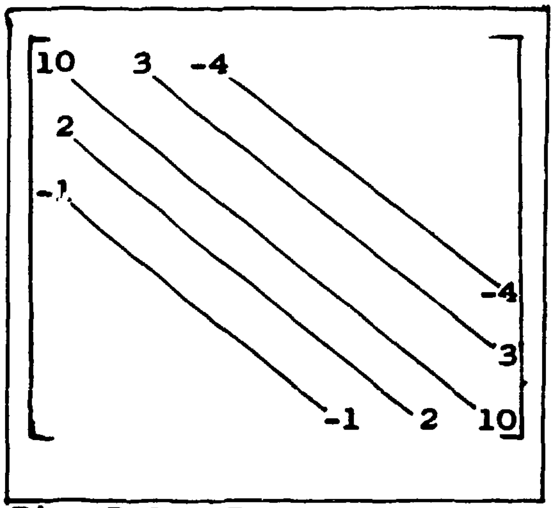


Fig. D-1. Test Matrix 1

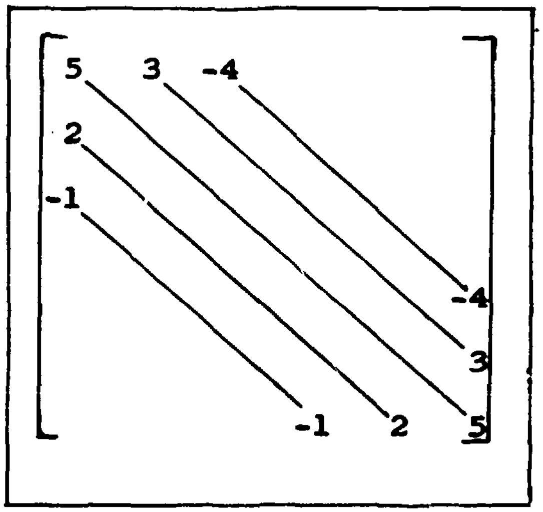


Fig. D-2. Test Matrix 2

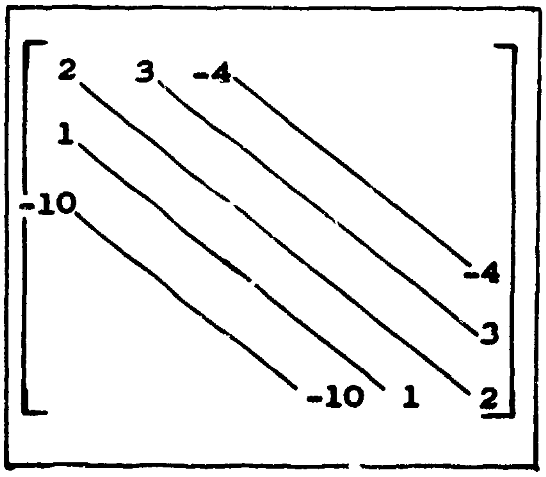


Fig. D-3. Test Matrix 3

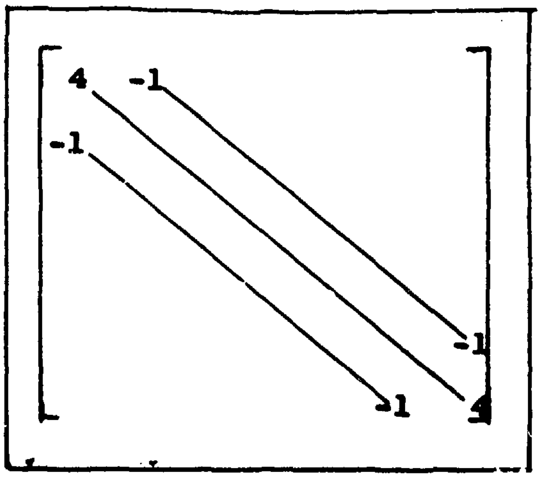


Fig. D-4. Test Matrix 4

APPENDIX E

Intermediate 'Test Matrices

Appendix E

Intermediate Test Matrices

This appendix contains the descriptions of each of the next eight test matrices and tables of performance for five sparse solvers.

Descriptions of Intermediate Test Matrices

Test Matrices five through twelve were all of size $n = 100$, with non-zeros at 3 to 5% sparsity. All values a_{ij} and the elements of $\underline{b}$ for each matrix were randomly generated by a standard function in the CDC 6600 computer. In some cases, the coordinates of the a_{ij} values were generated randomly; thus, not only were random values tested, but also random structures.

Test Matrices 5,6,7 - Three different tridiagonal matrices.

Test Matrix 8 - An arbitrary tridiagonal matrix with an additional 2% non-zero structure arbitrarily placed within ±5 diagonals of the main diagonal.

Test Matrix 9 - Similar to Test Matrix 8 except the 2% extra non-zeros are contained within ±10 diagonals.

Test Matrices 10,11 - Similar to Test Matrix 8 except that the 2% extra non-zeros are contained within ±15 diagonals.

Test Matrix 12 - Similar to Test Matrix 8 except that the 2% extra non-zeros are arbitrarily assigned throughout the entire matrix structure.

Performance Tables

The measured criteria contained in Tables E-I through E-V are the order of magnitude of the average scalar residual error (Eq [29]), the execution time, the number of times a fill-in value was generated, and the number of elements which the Gaussian algorithm deleted. In the case of Gaussian Elimination algorithms, the normalized pivot elements were also deleted because they were understood to be exactly 1.0 in value.

Table E-I
Intermediate Tests with SIMULT

Test Matrix	Error	Fill-ins	Deletions	Time (Seconds)
# 5	10^{-2}	97	295	0.26
# 6	10^{-4}	97	295	0.27
# 7	10^{-1}	97	295	0.30
# 8	10^{-8}	336	734	0.33
# 9	10^{-9}	987	1385	0.66
#10	10^{-8}	871	1269	0.64
#11	10^{-10}	1247	1645	0.89
#12	10^{-11}	2168	2566	1.70

ZTEST = 10^{-10}

Table E-II
Intermediate Tests with MFP2 (Partial Pivoting)

Test Matrix	Error	Fill-ins	Deletions	Time (Seconds)
# 5	10^{-14}	59	258	0.36
# 6	10^{-14}	61	260	0.39
# 7	10^{-13}	58	257	0.36
# 8	10^{-12}	291	512	1.02
# 9	10^{-12}	699	721	1.82
#10	10^{-13}	930	874	2.19
#11	10^{-13}	1044	946	2.46
#12	10^{-13}	4038	2405	21.11

Table E-III
Intermediate Tests with MFP3 (Full Pivoting)

Test Matrix	Error	Fill-ins	Deletions	Time (Seconds)
# 5	10^{-1}	232	305	0.73
# 6	10^{-1}	233	305	0.76
# 7	10^{+1}	218	302	0.74
# 8	10^{0}	1149	827	3.40
# 9	10^{0}	2002	1267	7.77
#10	10^{0}	2153	1105	7.79
#11	10^{0}	2129	1409	9.17
#12	10^{+1}	3815	2177	26.82

Table E-IV
Intermediate Tests with MFP4 (Min Row/Min Col)

Test Matrix	Error	Fill-ins	Deletions	Time (Seconds)
# 5	10^{-14}	0	199	0.37
# 6	10^{-14}	0	199	0.37
# 7	10^{-13}	0	199	0.43
# 8	10^{-1}	207	464	1.00
# 9	10^{0}	792	936	2.38
#10	10^{+4}	951	958	3.06
#11	failed - computed an infinite operand			
#12	10^{+1}	2117	1750	9.55

Table E-V
Intermediate Rests with MFP5 (Min Row/Max Element)

Test Matrix	Error	Fill-ins	Deletions	Time (Seconds)
# 5	10^{-14}	59	258	0.44
# 6	10^{-14}	61	260	0.45
# 7	10^{-13}	58	257	0.45
# 8	10^{-1}	249	504	1.03
# 9	10^{0}	629	771	1.90
#10	10^{+1}	932	1051	2.79
#11	10^{0}	717	779	2.10
#12	10^{0}	1713	1452	6.35

APPENDIX F

Test Matrices with Flanking Diagonals

Appendix F

Test Matrices with Flanking Diagonals

The following matrices (Figs. F-1 through F-8) were used in the final test phase of this thesis.

Specifications

In all cases, the rank, n, was 100, and all elements of the constant vector, b, were unity.

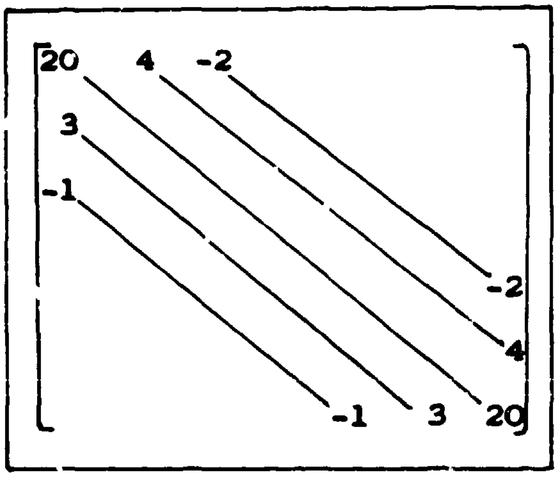


Fig. F-1. Test Matrix 13

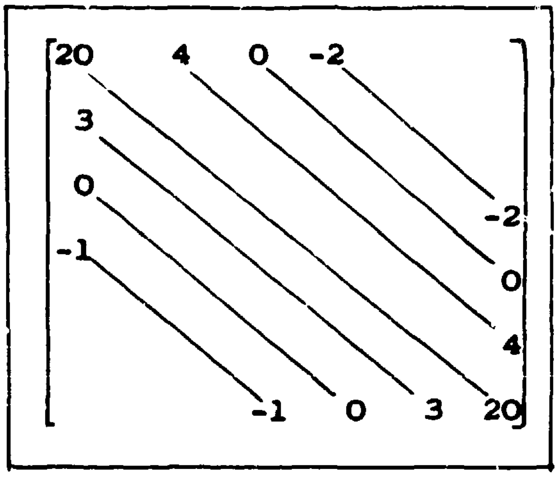


Fig. F-2. Test Matrix 14

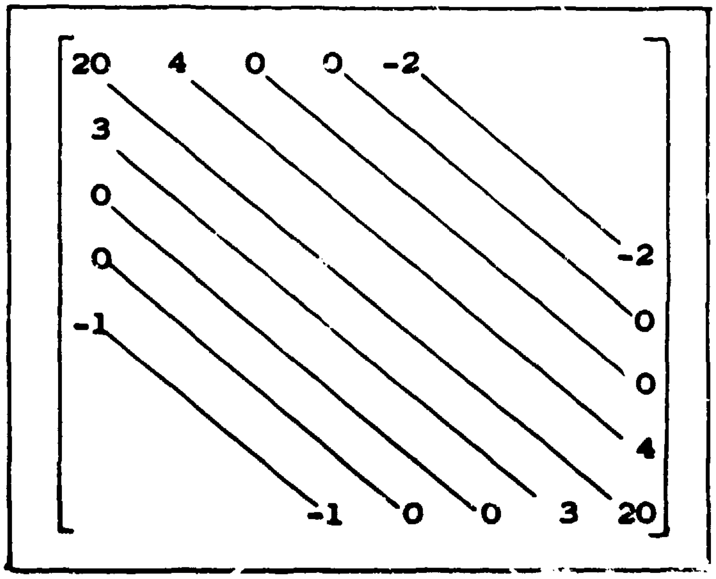


Fig. F-3. Test Matrix 15

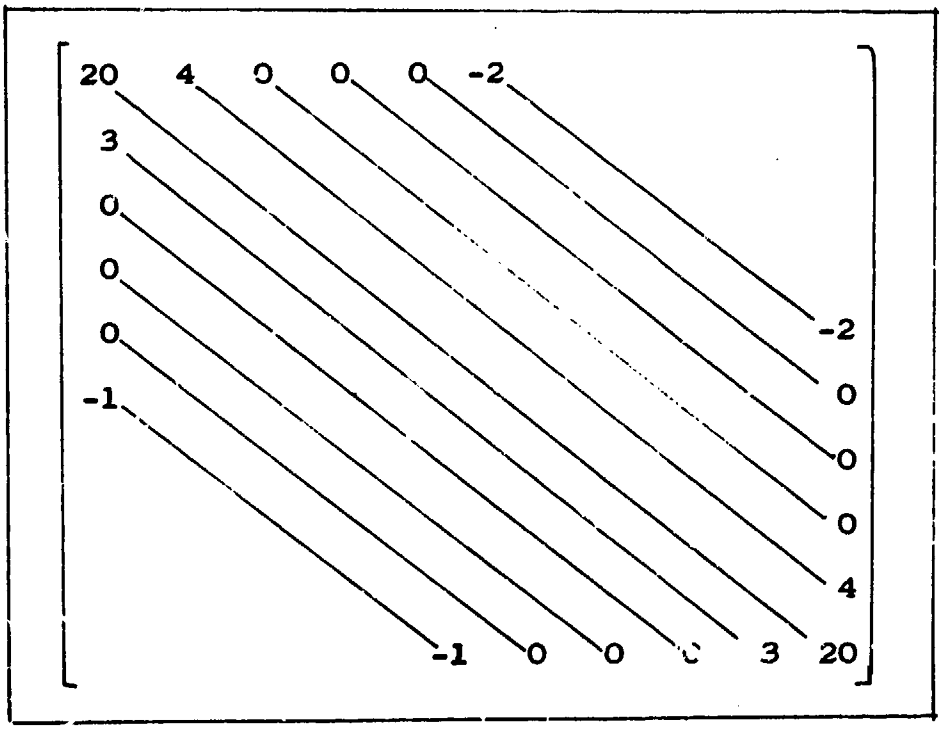


Fig. F-4. Test Matrix 16

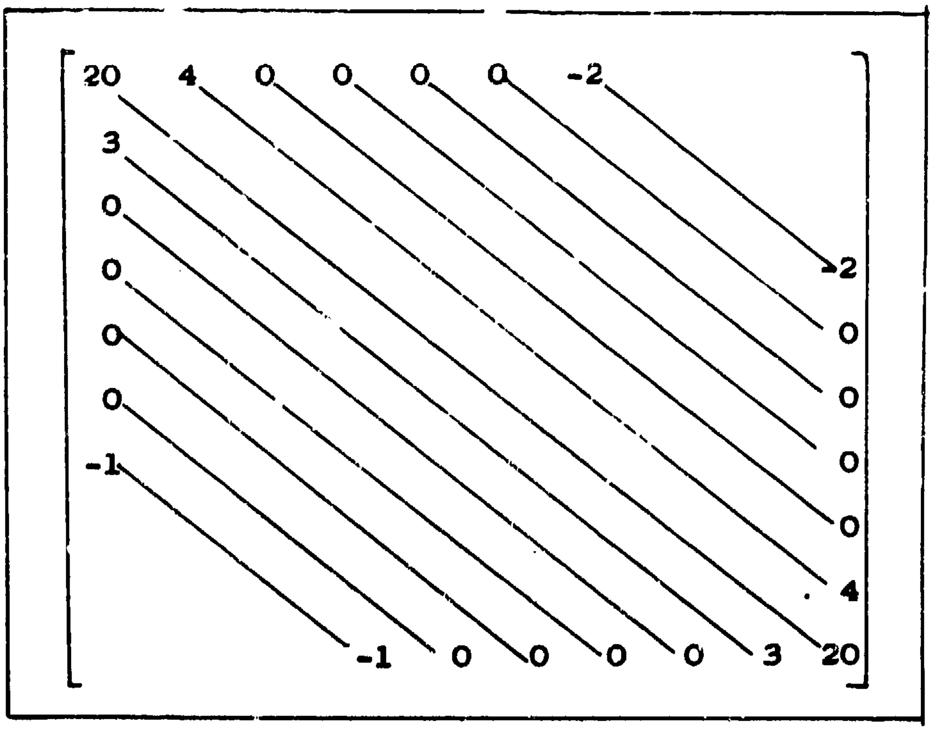


Fig. F-5. Test Matrix 17

$$\begin{bmatrix}
20 & 4 & 0 & 0 & 0 & 0 & 0 & -2 & & \\
3 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & -2 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
-1 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 4 \\
 & & -1 & 0 & 0 & 0 & 0 & 0 & 3 & 20
\end{bmatrix}$$

Fig. F-6. Test Matrix 18

$$\begin{bmatrix}
20 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & \\
3 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & -2 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
-1 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 4 \\
 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 20
\end{bmatrix}$$

Fig. F-7. Test Matrix 19

$$\begin{bmatrix}
20 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & & \\
3 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \\
0 & \ddots & & & & & & & & & \ddots & \\
0 & \ddots & & & & & & & & & & -2 \\
0 & \ddots & & & & & & & & & & 0 \\
0 & \ddots & & & & & & & & & & 0 \\
0 & \ddots & & & & & & & & & & 0 \\
0 & \ddots & & & & & & & & & & 0 \\
0 & \ddots & & & & & & & & & & 0 \\
-1 & \ddots & & & & & & & & & & 0 \\
 & & \ddots & & & & & & & & & 0 \\
 & & & & & & & & & & \ddots & 4 \\
 & & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 20
\end{bmatrix}$$

Fig. F-8. Test Matrix 20

APPENDIX G

Gaussian Elimination Program Listing

Appendix G

Gaussian Elimination Program Listing

This Appendix contains the listing of the Gaussian Elimination Sparse Solver called "SMART." This particular program is not the production model; the reader will note that the sparse matrix is read in from a permanent disk file called "TAPE2." The only significant difference between this listing and the listing of the production model is that all references to logical unit 2 have been replaced by list-directed "READ" commands from data cards.

```
      PROGRAM SMART(INPUT=/80,OUTPUT,TAPE3=OUTPUT,TAPE6,TAPE2)
C
C***********************************************************************
C
C     THIS PROGRAM IS THE FINAL RESULT OF AFIT THESIS GEO/PH/77-3
C     BY CAPT. MIKE POORE (GEO-77D).
C          THIS PROGRAM SOLVES A SPARSE MATRIX BY GAUSSIAN ELIM-
C          INATION WITH STRATEGIC PIVOTING.
C
C                THE MATRIX EQUATION IS AX=B
C
C
C*****************************VARIABLES*********************************
C
C                  ****************************
C                  * INTEGER ARRAY - INTEG(K) *
C                  ****************************
C
C         FIRST N+1 LOCATIONS: 3 X 20 BIT PARTIAL WORDS
C                      JSTAT    ISTAT    IA
C          JSTAT = NUMBER OF NON-ZEROS IN COLUMN K.
C          ISTAT = NUMBER OF NON-ZEROS IN ROW K.
C             IA = THE LOCATION OF THE START OF ROW K.
C
C          INTEG(N+2): KCOUNT -- THE INITAL COUNT OF NON-ZEROS.
C
C          NEXT N LOCATIONS: 3 X 20 BIT PARTIAL WORDS
C                      JCOL     IPIV     JPIV
C          JCOL  = A WORKING VECTOR FOR PIVOTING SCHEMES.
C          IPIV  = THE ORDER IN WHICH ROWS WERE USED TO PIVOT.
C          JPIV  = THE ORDER IN WHICH COLUMNS WERE USED TO PIVOT.
C
C                        *************
C                        ***CAUTION***
C                        *************
```

```
C
C           THE MAXIMUN NUMRER OF FOUATIONS WHICH THIS
C             RIT-SLICED CONFIGURATION CAN HANDLE IS
C                          524,287.
C
C                    *************************
C                    * REAL ARRAY - REALS(K) *
C                    *************************
C
C           FIRST N LOCATIONS:  THE B VECTOR
C
C           NEXT KCOUNT LOCATIONS:  THE ROW-RY-ROW PACK OF A
C                 NOTE: THE LOCATION CONTAINING AN A(I,J) VALUE
C                        IS SPLIT SO AS TO HAVE THE EXPONENT AND
C                       MANTISSA OF A(I,J) IN THE FIRST 50 BITS
C                       AND THE "J" INDEX IN THE LAST 10 BITS.
C
C                         *************
C                         ***CAUTION***
C                         *************
C
C            FOR PROBLEMS WHERE N IS GREATER THAN 1023,
C            THE MASKING CONSTANTS-IOCTAL(4) AND (5)-MUST
C            RE CHANGED ACCORDINGLY. USE OCTAL NUMBERS!
C
C***********************OCTAL CONSTANTS*******************************
C
C            IOCTAL(1) = A NUMRER WHICH WHEN ADDED TO INTEG(K)
C                        WILL INCREMENT ISTAT(K).
C            IOCTAL(2) = A NUMRER WHICH WHEN ADDED TO INTEG(K)
C                        WILL INCREMENT JSTAT(K).
C            IOCTAL(3) = A MASK FOR THE LEAST SIGNIFICANT 20
C                        BITS. WHEN INTEG(K) IS RIGHT SHIFTED
C                        20 BITS AND MASKED WITH IOCTAL(3) THE
C                        RESULT IS ISTAT.(40 BITS FOR JSTAT)
```

```
C            IOCTAL(4) = A MASK FOR REALS(K) WHICH WILL GIVE J.              000360
C                        J IS THE COLUMN NUMBER FOR THE REAL                 000365
C                        VALUE STORED IN REALS(K).                           000370
C            IOCTAL(5) = A MASK FOR THE VALUE OF A(I,J) STORED               000375
C                        IN THIS LOCATION.                                   000380
C                                                                            000385
C*****************************OTHER VARIABLES***************************000390
C                                                                            000395
C                                                                            000400
C               N= NUMBER OF EQUATIONS.                                      000405
C               X= THE SOLUTION VECTOR.                                      000410
C            IOPT= OPTION CODE FOR PIVOT SELECTION.                          000415
C           IVETO= PREVENTS RE-STARTING AFTER IOPT=1 FAILURE.                000420
C                                                                            000425
C***********************************************************************000430
C                                                                            000435
      COMMON INTEG(202),INTEGD,REALS(3600),IREALD,N,X(100),IOCTAL(5)         000440
      DIMENSION ITITLE(8)                                                    000445
      EXTERNAL GAUSSBX,THINKER,APRIORI                                       000450
      DATA ISTOP,ICHANGE/1HY,1HN/                                            000455
      REWIND 2                                                               000460
      REWIND 6                                                               000465
      IWR=8                                                                  000470
      INTEGD=202                                                             000475
      IREALD=3600                                                            000480
      IOCTAL(1)=4000000B                                                     000485
      IOCTAL(2)=20000000000000B                                              000490
      IOCTAL(3)=1777777B                                                     000495
      IOCTAL(4)=1777B                                                        000500
      ISKIP=0                                                                000510
C                                                                            000515
      IOCTAL(5)=77777777777777776000B                                        000505
C     READ TITLE TEXT                                                        000520
C                                                                            000525
      READ 75,(ITITLE(J),J=1,8)                                              000530
```

```
   75 FORMAT(8A10)
C
C     READ THE PIVOT OPTION!
C
C                       ****************************************
C                       *IOPT = 1 WILL CHOOSE THE A PRIORI PIVOTER. *
C                       *         THIS IS PREFERRED FOR DIAGONALLY  *
C                       *         DOMINANT MATRICES.                *
C                       *IOPT = 2 WILL CHOOSE A STABILIZING PIVOTER.*
C                       *         THIS IS PREFERRED FOR INCREASED   *
C                       *         ACCURACY - - - BASED ON "PIVTOL". *
C                       ****************************************
C
      READ*,IOPT
      IF(IOPT.EQ.2) GO TO 760
C
C     READ THE VETO CODE IF IOPT=1.....
C
      READ 70,IVETO
   70 FORMAT(A1)
C
C     THE PIVOT TOLERANCE (PIVTOL) IS NEEDED BY THE PIVOT SCHEME.
C     IT IS READ FROM DATA!
C
C                       ***********************************************
C                       *A LARGE PIVOT TOLERANCE SHOULD BE USED IF FILL-IN*
C                       *IS NOT EXPECTED TO BE A PROBLEM.                *
C                       *A SMALL PIVOT TOLERANCE MIGHT BE USED IF ACCURACY*
C                       *MUST BE SACRIFICED FOR THE SAKE OF FILL-IN.     *
C                       *THE LARGER THE PIVTOL, THE MORE THE PIVOTER     *
C                       *WILL EMULATE GAUSSIAN PARTIAL PIVOTING.         *
C                       ***********************************************
C
  760 IF((IOPT.EQ.1).AND.(IVETO.EQ.ISTOP)) GO TO 761
      READ*,PIVTOL
```

```
C
C     IFLAG1 AND IFLAG2 WERE REQUIREF ON TAPE 2 FOR PREVIOUS STUDY.
C
  761 READ(2) IFLAG1,IFLAG2
C
C-----------------------------------------------------------------------
C
C     PACK THF DATA (NON-ZEROS ONLY) IN A ROW-BY-ROW GROUP.
C
      CALL PACK1
C
C     ONCE THF DATA IS PACKED, COPY IT ONTO SCRATCH PAD FOR FUTURE
C     CALCULATION. TAPE6 IS THE SCATCH PAD.
C
      WRITF(6) N
      IWAIT=N+2
      WRITE(6)   (INTEC(I),I=1,IWAIT)
      IWAIT=INTEG(N+2)+N
      WRITF(6)  (REALS(I),I=1,IWAIT)
C     THE FI?ST GAUSSIAN PASS WILL FORM THE UPPER TRIANGULAR MATRIX
C     WITH PIVOT ELEMENTS UNDERSTOOD TO BE IDENTICALLY 1.0.
C     USE OF THE "TIMER" FUNCTION WILL GIVE MACHINE TIME FOR SOLUTION.
C
      IF(IOPT.LE.1) TIME1=TIMER(1,APRIORI,PIVTOL)
   81 IF(IOPT.GE.2) TIME1=TIMER(1,THINKER,PIVTOL)
C
C     THE SFCOND PASS PACKSOLVES FOR X.
C
      TIME2=TIMER(1,GAUSSBX,PIVTOL)
      TIME=TIME1+TIME2
C
C-----------------------------------------------------------------------
C
C     RE-ENTFR THE DATA FROM THE SCRATCH PAD
C
```

```
      REWIND 6
      READ(6) N
      IWAIT=N+2
      READ(6)    (INTEG(I),I=1,IWAIT)
      IWAIT=INTEG(N+2)+N
      READ(6) (REALS(I),I=1,IWAIT)
C
C     CHECK FOR LISTING OF A AND B DESIRED.
C
      IF(ISKIP.EQ.1) GO TO 999
      READ*, LISTIT
      IF(LISTIT.EQ.0) GO TO 899
      WRITE(IWR,888)
  888 FORMAT(1H1)
      PRINT*, " MATRIX NON-ZERO LISTING"
      DO 996 L=1,N
            LENG=(SHIFT(INTEG(L),-20)).AND.IOCTAL(3)
            KK=INTEG(L).AND.IOCTAL(3)
      DO 995 M=1,LENG
            ICHECK=(REALS(KK)).AND.IOCTAL(4)
            A=(REALS(KK)).AND.IOCTAL(5)
            WRITE(IWR,997) L,ICHECK,A
  997       FORMAT(7X,2HA(,I5,1H,,I5,3H)= ,1PE14.6)
            KK=KK+1
  995       CONTINUE
      B=REALS(L)
      WRITE(IWR,993) L,B
  993 FORMAT(30X,2HB(,I5,3H)= ,1PE14.6)
  996 CONTINUE
      WRITE(IWR,888)
      GO TO 999
  899 PRINT*, "MATRIX LISTING SUPPRESSED."
C
C---------------------------------------------------------------------------
C
```

```
C     ON A ROW-BY-ROW BASIS, MULTIPLY A BY X FOR THE RESIDUAL.
C
  999 RESID=0.
      DO 1 I=1,N
      CALL PMATMX(I,BPRIME)
      RESID=RESID + ABS(BPRIME-REALS(I))
    1 CONTINUE
C
C     WRITE OUT THE PERTINENT DATA.
C
      AVRES=RESID/N
C
C     IF THE AVG RESIDUAL IS GREATER THAN 0.01, RESET THE OPTION
C     CODE TO IOPT=2 AND TRY AGAIN IF NOT VETOED.
C
      IF((IOPT.EQ.1).AND.(AVRES.GT.0.01)) GO TO 80
C
C     IF IOPT=2 HAS ALREADY BEEN TRIED, WRITE A MESSAGE OF DIRECTION
C     AND ENCOURAGEMENT.
C
      IF(IVETO.NE.ICHANGE) GO TO 78
      IF(AVRES.GT.0.00001) WRITE(IWR,76) PIVTOL
   76 FORMAT(1X,///," PIVOT TOLERANCE IS ",F10.5,/," LARGER ONE MAY IMPR
     $OVE RESULTS.",/," USE WITH OPTION = 2 TO SAVE TIME.")
C
   78 WRITE(IWR,220)
      PRINT 74,(TITLE(M),M=1,8)
   74 FORMAT(26X,8A10)
      WRITE(IWR,222)
  222 FORMAT(1X,///)
      WRITE(IWR,230) N,AVRES
  200 FORMAT(1X,20HTHIS PROBLEM SOLVED ,I5,35H EQUATIONS WITH AVERAGE RE
     $SIDUAL = ,E17.10,/)
      PRINT*, "GAUSSIAN MATRIX SOLUTION TIME........."
      PRINT*, TIME
      READ*.IPIVOT
      IF(IPIVOT.EQ.0) GO TO 50
C
C     PRINT PIVOT SELECTION
C
      PRINT*, "                    PIVOT ORDERING"
      DO 666 K=1,M
      KJ=K+N+2
      IPIV=(SHIFT(INTEG(KJ),-20)).AND.IOCTAL(3)
      JPIV=INTEG(KJ).AND.IOCTAL(3)
      WRITE(IWR,665) IPIV,JPIV
  665 FORMAT(20X,I5,5X,I5)
  666 CONTINUE
  50  WRITE(IWR,220)
  220 FORMAT(1H1)
C
C     WRITE OUT X IN TABULAR FORM.
C
      CALL PRINTR(IWR,N,X)
      STOP "SMART"
C
C---------------------------------------------------------------------
C
  80  IF(IVETO.EQ.ISTOP) GO TO 78
      PRINT*,"OPTION CODE=1: RESIDUAL = ",AVRES
      PRINT*,"OPTION SHIFTED TO 2 AND PROBLEM RE-STARTED."
      IOPT=2
      ISKIP=1
      GO TO 81
C---------------------------------------------------------------------
      END
      SUBROUTINE PRINTR(IWR,N,X)
C*****************************************************************************
C
      DIMENSION X(N)
C
C     BLOCK DATA IN COLUMNS OF 50.........1,2,OR 3 COLUMNS PER PAGE
C     THE LAST FEW COLUMNS WILL NOT BE IN GROUPS OF 3, BUT AT LEAST THE
C     DATA WILL BE THERE IN AN ORGANIZED MANNER. THIS SCHEME IS DESIGNED
C     TO SAVF PAPER.
C
      ICOUNT=0
C     ICOUNT IS THE NUMBER OF X(I),S PRINTED SO FAR.
C
      IPRINT=N
C     IPRINT IS THE NUMBER OF VALUES LEFT TO BE PRINTED.
C
   70 IF(IPRINT.LT.150) GO TO 9
      IWRITE=ICOUNT + 1
      IIWRT=IWRITE+49
      DO 8 I=IWRITE,IIWRT
      II=I+50
      III=I+100
      WRITF(IWR,80) I,X(I),II,X(II),III,X(III)
   80 FORMAT(15X,3(2HX(,I5,3H)= ,1PE18.10,10X))
    8 CONTINUE
      IPRINT=IPRINT-150
      ICOUNT=ICOUNT+150
      WRITE(IWR,800)
  800 FORMAT(1H1)
      GO TO 70
    9 IF(IPRINT.LT.100) GO TO 10
      IWRITE= ICCUNT +1
      IIWRT=IWRITE+49
      DO 90 I=IWRITE,IIWRT
      II=I+50
      WRITE(IWR,95) I,X(I),II,X(II)
   95 FORMAT(15X,2(2HX(,I5,3H)= ,1PE18.10,10X))
   90 CONTINUE
      IPRINT=IPRINT-100
      ICOUNT=ICOUNT+100
      WRITE(IWR,600)
   10 IF(IPRINT.LT.50)  GO TO 15
      IWRITE=ICOUNT+1
      IIWRT=IWRITE+49
      DO 91 I=IWRITE,IIWRT
      WRITE(IWR,92) I,X(I)
   92 FORMAT(15X,2HX(, I5, 3H)=  ,1PE18.10)
   91 CONTINUE
      IPRINT=IPRINT-50
      ICOUNT=ICOUNT+50
   15 IWRITE=ICOUNT+1
      IF((ICOUNT.EQ.N).OR.(IPRINT.EQ.0)) RETURN
      DO 30 I=IWRITE,N
      WRITE(IWR,92) I,X(I)
   30 CONTINUE
      RETURN
C-----------------------------------------------------------------------
      END
      SUBROUTINE PACK1                                                  001680
C***********************************************************************001685
C                                                                       001690
      COMMON INTEG(202),INTEGD,REALS(3600),IREALD,N,X(100),IOCTAL(5)    001695
C                                                                       001700
C     THIS PROGRAM LOADS THE SPARSE MATRIX FOR SMART.                   001705
C                                                                       001710
C         KCOUNT = THE NUMBER OF NON-ZERO ELEMENTS ALREADY LOADED AT    001715
C                  ANY POINT.                                           001720
C                                                                       001725
C         KSTRT  = THE STARTING ADDRESS OF THE NEXT ROW.                001730
C                                                                       001735
C    *******************************************************************001740
C    *IMPORTANT.......ONE CAN SEGMENT ROWS IF DESIRED FOR ANY ROW      *001745
C    *INCLUDING THE LAST ROW. TO SWITCH CONTROL TO READ IN 8, THE LAST *001750
C    *ROW INDEX DATA CARD MUST BE "NUMBER,0,0" WHERE "NUMBER"= ANY     *001755
C    *INTEGER GREATER THAN N. FAILURE TO INSTALL THIS TRAILER CARD WILL*001760
C    *CAUSE A LIST DIRECTED READ ERROR.                                *001765
C    *******************************************************************001770
C                                                                       001775
      NMAX=(INTEGD-2)/2                                                 001780
      READ(2) N                                                         001785
      IF(N.GT.NMAX) STOP "TOO MANY ROWS"                                001790
      KCOUNT=0                                                          001795
      KSTRT=N+1                                                         001800
      I=0                                                               001805
C                                                                       001810
C     ZERO OUT AVAILABLE STORAGE SPACE                                  001815
C                                                                       001820
      DO 69 IBL=1,INTEGD                                                001825
   69 INTEG(IBL)=0                                                      001830
      DO 68 IBL=1,IREALD                                                001835
   68 REALS(IBL)=0.                                                     001840
C                                                                       001845
C     DATA FORMAT REQUIRES THE ROW NUMBER, THE COORDINATE OF THE        001850
```

```
C     LEFTMOST COLUMN, AND THE LENGTH OF THE FOLLOWING NON-ZERO
C     ELEMENTS.
C
  1   READ(2)  INEW,JLEFT,JLENG
      IF((JLEFT.EQ.0).OR.(JLENG.EQ.0)) GO TO 99
      JEXTPA=JLEFT+JLENG-1
C
C     CHECK FOR BOUNDARY ON RIGHT EDGE OF MATRIX
C
      IF(JEXTRA.GT.N) STOP "BAD ROW EXTEND"
      IF(INEW.NE.I+1) GO TO 10
      IF(INEW.GT.N) GO TO 99
C
C     IS THE NEXT CARD THE BEGINNING OF THE NEXT ROW OR A SEGMENT EX-
C     TENSION OF THE PRESENT ROW..........
C
C     NEW ROW......
C
      JINDEX=0
      I=I+1
C
C     LOG IN  THE BEGINNING OF THE NEW ROW
C
      IA=KSTRT
      INTEG(I)=IA
      GO TO 15
C
C     SAME ROW....AN EXTENSION THEREOF....
C
 10   IF(INEW.EQ.I) GO TO 11
      IF(INEW.GT.N) GO TO 99
      STOP "ROW OUT OF SEQ"
C
C     CHECK FOR OVERLAP OF DATA
C
```

```
   11 IF(JINDEX.GT.JLEFT) STOP "ROW OVERLAP"
C
C     READ IN DATA
C
   15 MAX=KSTRT+JLENG-1
      KCOUNT=KCOUNT+JLENG
      READ(2) (REALS(M),M=KSTRT,MAX)
      DO 5 M=KSTRT,MAX
      INTEG(I)=INTEG(I)+IOCTAL(1)
      REALS(M)=(REALS(M).AND.IOCTAL(5)).OR.JLEFT
      JLEFT=JLEFT+1
    5 CONTINUE
      JINDEX=JLEFT
C
C     THE ABOVE BUILDS THE WORKING SPACE WITH BIT MANIPULATION.
C     NOTE HOW THE ROW AND COLUMN STATUS VECTORS ARE ALSO BUILT.
C
      N1 = KSTRT
      N2 = MAX-1
C
C     ASSIGN STARTING ADDRESS FOR NEXT ROW
C
      KSTRT=MAX+1
C
C     BUT ALSO CHECK FOR ZEROS WHICH MAY HAVE BEEN READ IN
C     NOTE THAT THE FIRST TIME WE DO A FLOATING POINT COMPARSION
C     FOR A NUMBER WHICH IS IDENTICALLY = 0.0, WE SHOULD GET A POSITIVE
C     CHECK.
C
      DO 50 M=N1,N2
      AM=REALS(M).AND.IOCTAL(5)
      IF(AM.NE.0.) GO TO 55
      J=REALS(M).AND.IOCTAL(4)
      INTEG(I)=INTEG(I)-IOCTAL(1)
      DO 49 MM=M,N2
      REALS(MM) = REALS(MM+1)
   49 CONTINUE
      KCOUNT=KCOUNT-1
      KSTRT=KSTRT-1
   55 CONTINUE
   50 CONTINUE
C
C     THE ABOVE SHUFFLES ENTIRE ARRAY UPWARDS ONE LOCATION.
C
C     READ NEXT CARD
C
      GO TO 1
C
C     ENTER LAST ROW STATUS VALUE
C
   99 INTEG(I+1)=KSTRT
C
      IF(I.LT.N) STOP "TOO FEW ROWS"
      IF(I.GT.N) STOP "TOO MANY ROWS"
C
C     NOW READ IN B VECTOR
C
      READ(2)   (REALS(M), M=1,N)
C
C     NOW TALLY UP THE NUMBER OF NON-ZEROS
C
      INTEG(N+2)=KCOUNT
C
      RETURN
C------------------------------------------------------------------------
      END
      SUBROUTINE RMATMX(I,BPRIME)
C***********************************************************************
C
      COMMON INTEG(202),INTEGO,REALS(3600),IREALO,N,X(100),IOCTAL(5)
C
C     THIS SUBROUTINE IS USED  TO CALCULATE THE RESIDUAL. USING MATRIX
C     MULTIPLICATION, THE ITH ROW OF THE ORIGINAL A MATRIX WILL BE
C     MULTIPLIED BY X. THE SCALAR SUM, BPRIME, WILL SUBSEQUENTLY BE
C     COMPARED TO B(I) IN THE MAIN PROGRAM.
C
      BPRIME=0.
C
C     ONLY MULTIPLY THE NON-ZEROS TO SAVE TIME
C
      ICOUNT=(SHIFT(INTEG(I),-20)).AND.IOCTAL(3)
C
C     WHAT IS THE STARTING ADDRESS OF THE LEFT-MOST NON-ZERO COLUMN.
C
      K=INTEG(I).AND.IOCTAL(3)
C
      DO 1 L=1,ICOUNT
C
C     WHAT IS THE COLUMN NUMBER OF THE NON-ZERO AND THE INDEX OF X(J)
C
      J=REALS(K).AND.IOCTAL(4)
C
C     MULTIPLY
C
      AK=REALS(K).AND.IOCTAL(5)
      BPRIME=BPRIME + AK*X(J)
C
C     GET NEXT NON-ZERO
C
      K=K+1
    1 CONTINUE
```

```
      RETURN                                                            002535
C-----------------------------------------------------------------------002540
      END                                                               002545
```

```
         IDENT   TIMER                                                        002550
         ENTRY   TIMER,TNULL                                                  002555
         TITLE   FUNCTION TIMER (NUM,FUNC,ARG1,ARG2,ARG3, ... ,ARGN)          002560
*                                                                             002565
*        ROUTINE BY HARRY M. MURPHY, JR.,  13 DECEMBER 1973.                  002570
*        THIS ROUTINE MUST BE CALLED BY FTN-COMPILED FORTRAN.                 002575
*                                                                             002580
         EXT     SECOND                                                       002585
*                                                                             002590
*        FUNCTION TIMER (NUM,FUNC,ARG1,ARG2,ARG3, ... ,ARGN)                  002595
+        VFD     42/5LTIMER,18/77B                                            007600
 TIMER   EQ      B0,B0,*+1S17                                                 002605
+        SB7     B0+1           .B7 = 1.                                      002610
         SA5     X1             .X5 = NUM.                                    002615
         SA2     A1+B7          .X2 = (FUNC).                                 002620
         BX4     X5             .X4 = NUM.                                    002625
         AX4     73B            .EXTEND SIGN BIT IN X4.                       002630
                                                                              002635
         MX3     17             .X3 = SHIFTED 377777.                         002640
         BX6     X4-X5          .X6 = ABS(NUM).                               002645
         LX3     17             .X3 = 2**17-1.                                002650
         NZ      X6,TM1         .IF NUM IS NON-ZERO, SKIP.                    002655
         SX6     B7             .OTHERWISE, SET NUM TO 1.                     002660
 TM1     IX4     X6-X3          .X4 = NUM - (2**17-1).                        002655
         NG      X4,TM2         .IF NUM LESS THAN (2**17-1), SKIP.            002670
         BX6     X3             .OTHERWISE, SET NUM = 2**17-1.                002675
 TM2     PX7     B0,X6          .X7 = FLOAT(NUM).                             002680
         SA6     NUM            .STORE X6 AS NUM.                             002685
         NX7     B0,X7          .NORMALIZE X7.                                002690
         MX0     42             .GET 42-BIT MASK IN UPPER X0.                 002695
         SA7     A6+B7          .STORE X7 AS FNUM.                            002700
         SX6     A2+B7          .X6 = ARGL.                                   002705
         SA6     A7+B7          .STORE X6 AS ARGL.                            002710
         SA1     A6+B7          .X1 = RJ INSTRUCTION.                         002715
         BX3     -X0*X2         .MASK ADDRESS TO X3.                          002720
         LX3    30          .SHIFT ADDRESS TO UPPER X3.                      002725
         BX6    X1+X3       .SPLICE RJ AND ADDRESS.                          002730
         SA6    CALL        .STORE NEW RJ INSTRUCTION IN CALL.               002735
         SX7    TZ          .X7 = (TZ).                                      002740
         SA7    APL         .STORE (TZ) IN APL.                              002745
         SA1    APL         .A1 = (APL).                                     002750
         RJ     SECOND      .CALL SECOND.                                    002755
*                                                                            002760
+        SA2    ARGL        .X2 = ARGL.                                      002765
         SA1    X2          .A1 = ARGL.                                      002770
*                                                                            002775
 CALL    RJ     *+1S17      .CALL FUNCT.                                     002780
+        SA2    NUM         .X2 = NUM.                                       002785
         SA3    ARGL        .X3 = ARGL.                                      002790
         SX6    X2-1        .DECREMENT NUM IN X6.                            002795
                                                                             002800
         SA1    X3          .A1 = ARGL.                                      002805
         SA6    A2          .RE-STORE DECREMENTED NUM.                       002810
         NZ     X6,CALL     .IF NUM NOT YET ZERO, LOOP TO CALL.              002815
*                                                                            002820
         SX7    TX          .X7 = (TX).                                      002825
         SA7    APL         .STORE (TX) IN APL.                              002830
         SA1    APL         .A1 = (APL).                                     002835
         RJ     SECOND      .CALL SECOND.                                    002840
+        SA1    TZ          .X1 = TZ.                                        002845
         SA2    FNUM        .X2 = FNUM.                                      002850
         FX3    X6-X1       .X3 = ELAPSED TIME IN SECONDS.                   002855
         NX3    B0,X3       .NORMALIZE X3.                                   002860
         FX4    X3/X2       .COMPUTE MEAN TIME PER CALL.                     002865
         SA5    =6.0E-6     .X5 = MEAN OVERHEAD TIME PER CALL.               002870
         FX6    X4-X5       .SUBTRACT MEAN OVERHEAD TIME.                    002875
         NX6    B0,X6       .NORMALIZE X6.                                   002880
         EQ     B0,B0,TIMER .AND EXIT.                                       002885
*                                                                            002890
*        TNULL, A DUMMY ROUTINE FOR ESTABLISHING OVERHEAD TIME.              002895
```

```
+          VFD    42/5LTNULL,18/0
 TNULL     EQ     B0,B0,*+1S17
+          EQ     B0,B0,TNULL
*
 NUM       BSS    1
 FNUM      BSS    1
 ARGL      BSS    1
 RJMP      PJ     0
 TZ        BSS    1
 TX        BSS    1
 APL       BSSZ   2
           END
```

```
      SUBROUTINE APRIORI(PIVTOL)
C***********************************************************************
C
      COMMON INTEG(202),INTEGO,REALS(3600),IREALD,N,X(100),IOCTAL(5)
C
      DUMMY=PIVTOL
C
C*********PIVOTING IS DIAGONAL OR FIRST NON-ZERO OF NEXT ROW***********
C
C     WORKING SPACE IS LOCATED IN INTEG(N+3) THRU INTEG(INTEGO) IN THE
C     FIRST 20 BITS OF THAT REGISTER.
C
C***********************************************************************
C
C     COMMENCE PIVOTING
C
      NN=N
      DO 1 K=1,NN
      IELROW=K
      J=K
      GO TO 643
  641 JMIN=INTEG(IELROW).AND.IOCTAL(3)
      J=REALS(JMIN).AND.IOCTAL(4)
C
C     FETCH PIVOT ELEMENT:
C
  643     KKL=INTEG(IELROW).AND.IOCTAL(3)
          KAV=KKL
          INDEXF=KKL-1
          LENGRW=(SHIFT(INTEG(IELROW),-20)).AND.IOCTAL(3)
          DO 12 M1=1,LENGRW
          JAIN=REALS(INDEXF+M1).AND.IOCTAL(4)
          IF(J.EQ.JAIN) GO TO 120
  12      CONTINUE
          GO TO 641
  120 PIVOT=PEALS(INDEXF+M1).AND.IOCTAL(5)
            IHOLD=INDEXF+M1
C
C     NORMALI7E PIVOT ROW:
C
  648 INDEXN=KAY+LENGRW-1
      DO 15 M1=KAY,INDEXN
      JNORM=REALS(M1).AND.IOCTAL(4)
      REALS(M1)=((REALS(M1)/PIVOT).AND.IOCTAL(5)).OR.JNORM
   15 CONTINUF
      REALS(IELROW)=REALS(IELROW)/PIVOT
C
C     DELETE PIVOT ELEMENT:
C
      MAX=(INTFG(N+1).AND.IOCTAL(3))-2
      DO 13 M1=IHOLD,MAX
      PEALS(M1)=REALS(M1+1)
   13 CONTINUF
      L1=IELROW+1
      L2=N+1
      DO 14 M1=L1,L2
      INTEG(M1)=INTEG(M1)-1
   14 CONTINUF
      INTEG(IFLROW)=INTEG(IELROW)-IOCTAL(1)
      LENGRW=LENGRW-1
C
C     UPDATF PIVOT VECTOR INFORMATTION:
C
      IREG=IFLROW
      IREG=(SHIFT(IREG,20))+ J
      INTEG(K+N+2)=IREG
      INTEG(IELROW)=INTEG(IELROW) + IOCTAL(2)
C
C     IF THE LAST ROW/COLUMN COORDINATES HAVE BEEN LOADED INTO
C     IPIV AND JPIV, THERE ARE NO MORE ROWS INTO WHICH WE CAN
```

```
C     SUBSTITUTE. EXIT THIS "DO LOOP".
C
      IF(K.EQ.N) GO TO 20
C
C-------------------------------------------------------------------------
C
C     OTHERWISE, WHAT ARE THE ROWS WHICH HAVE NOT YET BEEN USED IN
C     PIVOTING AND HAVE NON-ZEROS IN THE JTH COLUMN TO BE ELIMINATED?
C
      IROWS=0
      KKK=1
      IRSTRT=K+1
          DO 2 I=IRSTRT,N
          ICHECK=INTEG(I).AND.IOCTAL(3)
          LENG=(SHIFT(INTEG(I),-20)).AND.IOCTAL(3)
C
               DO 3 L=1,LENG
C
C     ONCE THE CHECK IS MADE FOR JAICK .GT. J, IT IS ASSUMED
C     THE VALUE WILL NOT BE FOUND.
C
               JAICK=REALS(ICHECK).AND.IOCTAL(4)
               IF(JAICK.GT.J) GO TO 2
               IF(JAICK.NE.J) GO TO 30
               JCOLL=I
               JCOL=SHIFT(JCOLL,40)
               NRD=KKK+N+2
               INTEG(NRD)=(INTEG(NRD).AND.(.NOT.MASK(20)))+JCOL
               IROWS=IROWS+1
               KKK=KKK+1
               GO TO 2
   30          ICHECK=ICHECK+1
   3           CONTINUE
   2      CONTINUE
      IF(IROWS.LE.0) GO TO 1
```

```
C
C     NOW THAT THE TALLY OF ROWS IS DONE, SUBSTITUTE THE EQUIVALENT
C     PIVOT VALUE INTO EACH ROW....COLUMN FOR COLUMN.  OBSERVE FILL-IN.
C
          DO 4 M=1,IROWS
          ISUB=(SHIFT(INTEG(N+2+M),-40)).AND.IOCTAL(3)
C
C     FETCH SUBSTITUTIONAL ELEMENT:
C
      LENGSUB=(SHIFT(INTEG(ISUB),-20)).AND.IOCTAL(3)
      KKL=INTEG(ISUB).AND.IOCTAL(3)
      KLIP=KKL
      INDEXS=KKL-1
      DO 16 M1=1,LENGSUB
      JAIN=REALS(INDEXS+M1).AND.IOCTAL(4)
      IF(J.EQ.JAIN) GO TO 160
   16 CONTINUE
      FACTOR=0.0
      GO TO 161
  160 FACTOR=REALS(INDEXS+M1).AND.IOCTAL(5)
      IHOLD=INDEXS+M1
C
C     SINCE THIS LOCATION IS BEING SUBSTITUTED FOR, DELETE IT. IT WILL
C     SOON EQUAL ZERO
C
  161 MAX=(INTEG(N+1).AND.IOCTAL(3))-2
      DO 17 M3=IHOLD,MAX
      REALS(M3)=REALS(M3+1)
   17 CONTINUE
      L1=ISUB+1
      L2=N+1
      DO 18 M3=L1,L2
      INTEG(M3)=INTEG(M3)-1
   18 CONTINUE
      INTEG(ISUB)=INTEG(ISUB)-IOCTAL(1)
```

```
C
C     THE NUMBER OF CALCULATIONS IS THE NUMBER OF NON-ZEROS LEFT
C     EXCEPT THE PIVOT ELEMENT WHICH HAS ALREADY BEEN DELETED.
C
C-----------------------------------------------------------------------
C
C     RE-COMPUTE CONSTANT TERM.
C
         REALS(ISUB)=REALS(ISUB)-REALS(IELROW)*FACTOR
C
C     NOW WORK ON THE A MATRIX VALUES NOTING THE FILL-IN
C
      ISTRT=INTEG(IELROW).AND.IOCTAL(3)
               DO 5 L=1,LENGRW
               JJ=REALS(ISTRT).AND.IOCTAL(4)
               XFER=(REALS(ISTRT).AND.IOCTAL(5))*FACTOR
C
C     SEE IF A JTH COLUMN EXISTS IN ROW "ISUB"
C
      LENGSUB=(SHIFT(INTEG(ISUB),-20)).AND.IOCTAL(3)
      KGB=INTEG(ISUB).AND.IOCTAL(3)
      INDEXF=KGB-1
      DO 64 M9=1,LENGSUB
      JAIN=REALS(INDEXF+M9).AND.IOCTAL(4)
      IF(JJ.EQ.JAIN) GO TO 664
   64 CONTINUE
      VALUE=0.
      IHOLD=-1
      GO TO 604
  664 VALUE=REALS(INDEXF+M9).AND.IOCTAL(5)
      IHOLD=INDEXF+M9
  604 VALNEW=VALUE-XFER
C
C     IF THERE IS A VALUE, COMPUTE IT. IF NOT, FILL-IN.
C
```

```
      IF(IHOLD.LE.0) GO TO 200
      REALS(IHOLD)=(VALNEW.AND.IOCTAL(5)).OR.JJ
      GO TO 201
C
C     FILL IN:
C
  200 KLIP=INTEG(ISUB).AND.IOCTAL(3)
      JAK=PFALS(KLIP).AND.IOCTAL(4)
      LENGFIL=(SHIFT(INTEG(ISUB),-20)).AND.IOCTAL(3)
      JAKL1=PEALS(KLIP+LENGFIL-1).AND.IOCTAL(4)
      IF(JJ.LT.JAK) GO TO 50
      IF(JJ.GT.JAKL1) GO TO 51
      MFIN=(TNTEG(ISUB+1).AND.IOCTAL(3))-1
C
C     FIND BOUNDS ON JJ:
C
      DO 222 M4=KLIP,MFIN
      JAM=PEALS(M4).AND.IOCTAL(4)
      JAM1=REALS(M4+1).AND.IOCTAL(4)
      IF((JJ.GT.JAM).AND.(JJ.LT.JAM1)) GO TO 52
  222 CONTINUE
      IAI1=INTEG(ISUB+1).AND.IOCTAL(3)
      IF(KLIP.NE.IAI1) STOP "BAD FILL-IN"
      IBUMPS=IAI1
      GO TO 60
   50 IBUMPS=KLIP
      GO TO 60
   51 IBUMPS=INTEG(ISUB+1).AND.IOCTAL(3)
      GO TO 60
   52 IBUMPS=M4+1
   60 IAN1=INTEG(N+1).AND.IOCTAL(3)
      IBMPLN=IAN1-IBUMPS
      DO 70 L4=1,IBMPLN
      INDFILL=IAN1-L4
      REALS(INDFILL+1)=REALS(INDFILL)
   70 CONTINUE
C
C     ASSIGN FILL-IN ITS NEW LOCATION:
C
      REALS(IBUMPS)=(VALNEW.AND.IOCTAL(5)).OR.JJ
C
C     RE-ARRANGE COORDINATE TABLES
C
      INI=ISUB+1
      NIN=N+1
      DO 80 L5=INI,NIN
      INTEG(L5)=INTEG(L5)+1
   80 CONTINUE
C
C     UPDATE STATUS VECTOR
C
      INTEG(ISUB)=INTEG(ISUB)+IOCTAL(1)
      IAN1=INTEG(N+1).AND.IOCTAL(3)
      IF(IAN1.GE.IREALD) STOP "TOO MUCH FILL-IN"
  201 ISTRT=ISTRT+1
    5 CONTINUE
    4 CONTINUE
    1 CONTINUE
   20 ICHECK=(SHIFT(INTEG(2+2*N),-20)).AND.IOCTAL(3)
C
C     MAKE SURE LAST PIVOT HAS BEEN ELIMINATED:
C
      ISTT=(SHIFT(INTEG(ICHECK),-20)).AND.IOCTAL(3)
      IF(ISTT.GT.0) STOP "LAST PIVOT BAD"
      RETURN
C-----------------------------------------------------------------------
      END
      SUBROUTINE THINKER(PIVTOL)                                        004175
C***********************************************************************004180
C                                                                       004185
      COMMON INTEG(202),INTEGD,REALS(3600),IREALD,N,X(100),IOCTAL(5)    004190
C                                                                       004200
C     THIS SUBROUTINE DOES THE FORWARD  GAUSSIAN ELIMINATION. ELEMENTS  004210
C     WILL BE MANIPULATED SO AS TO LEAVE AN UPPER TRIANGULAR MATRIX WITH004215
C     ALL PIVOT ELEMENTS =1.                                            004220
C     NOTE THAT WITH PIVOT STRATEGY, THE MATRIX WILL NOT APPEAR         004225
C      TO BE UPPER TRIANGULAR SINCE THE ROWS WILL BE ORDERED            004230
C      WITH IPIV AND JPIV.                                              004235
C                                                                       004240
C*************PIVOTING IS CONSECUTIVELY CALCULATED***********************004245
C                                                                       004250
C     WORKING SPACE IS LOCATED IN INTEG(N+3) THRU INTEG(INTEGD) IN THE  004255
C     FIRST 20 BITS. THE COLUMN STATUS VECTOR IS LOCATED IN INTEG(1)    004260
C     THRU INTEG(N) IN THE FIRST 20 BITS.                               004265
C                                                                       004270
C***********************************************************************004275
C                                                                       004280
C     CONSTRUCT THE INITIAL COLUMN STATUS VECTOR                        004285
C                                                                       004290
      ISTOP=INTEG(N+2)                                                  004295
      DO 606 I=1,ISTOP                                                  004300
      INDEX=I+N                                                         004305
      JJJ=REALS(INDEX).AND.IOCTAL(4)                                    004310
      INTEG(JJJ)=INTEG(JJJ)+IOCTAL(2)                                   004315
  606  CONTINUE                                                         004320
C                                                                       004325
C     COMMENCE PIVOTING                                                 004330
C                                                                       004335
      NN=N                                                              004340
      DO 1 K=1,NN                                                       004345
      IELROW=K                                                          004350
C                                                                       004355
```

```
C     FINO THF COLUMN WITH THE LEAST NUMBER OF NON-ZEROS.                004360
C                                                                        004365
      LENGPW=(SKIFT(INTEG(IELROW),-20)).ANO.IOCTAL(3)                   004370
      KK=INTEG(IELROW).ANO.IOCTAL(3)                                    004375
      KKL=KK                                                            004380
      LENGCL=N                                                          004385
           DO 25 M=1,LENGRW                                             004390
           JAKK=REALS(KK).ANO.IOCTAL(4)                                 004395
           JTEST=(SHIFT(INTEG(JAKK),-40)).ANO.IOCTAL(3)                 004400
           IF(JTEST.GE.LENGCL) GO TO 250                                004405
           MINCCL=JAKK                                                  004410
           LENGCL=JTEST                                                 004415
  250      KK=KK+1                                                      004420
  25       CONTINUE                                                     004425
      J=MINCOL                                                          004430
C                                                                        004435
C     FETCH PIVCTAL ELEMENT! IF IT IS BELOW THE PIVOT TOLERANCE          004440
C     THEN, INSTEAD, USE THE LARGEST ABSOLUTE VALUE IN THE ROW           004445
C     TO DRIVF THE SYSTEM TO MORE STABILITY.                             004450
C                                                                        004455
      KAY=KKL                                                           004460
      INDEXF=KKL-1                                                      004465
      DO 12 M1=1,LENGRW                                                 004470
      JAIN=REALS(INDEXF+M1).ANO.IOCTAL(4)                               004475
      IF(J.EQ.JAIN) GO TO 120                                           004480
  12  CONTINUE                                                          004485
      GO TO 300                                                         004490
  120 PIVOT=REALS(INDEXF+M1).ANO.IOCTAL(5)                              004495
      IHOLD=INDEXF+M1                                                   004500
      IF(ABS(PIVOT).LT.PIVTOL) GO TO 300                                004505
C                                                                        004510
C     OTHERWISE, NORMALIZE THE ROW AND DELETE THE PIVOT                  004515
C                                                                        004520
C     NORMALIZE ROW:                                                     004525
C                                                                        004530
```

```
  410 INDEXN=KAY+LENGRW-1                                             004535
      DO 15 M1=KAY,INDEXN                                             004540
      JNORM=REALS(M1).AND.IOCTAL(4)                                   004545
      REALS(M1)=((REALS(M1)/PIVOT).AND.IOCTAL(5)).OR.JNORM            004550
   15 CONTINUE                                                        004555
      REALS(IELROW)=REALS(IELROW)/PIVOT                               004560
C                                                                     004565
C     DELETE PIVOT ELEMENT                                            004570
C                                                                     004575
      MAX=(INTEG(N+1).AND.IOCTAL(3))-2                                004580
      DO 13 M1=IHOLD,MAX                                              004585
      REALS(M1)=REALS(M1+1)                                           004590
   13 CONTINUE                                                        004595
      L1=IELROW+1                                                     004600
      L2=N+1                                                          004605
      DO 14 M1=L1,L2                                                  004610
      INTEG(M1)=INTEG(M1) -1                                          004615
   14 CONTINUF                                                        004620
      INTEG(IFLROW)=INTEG(IELROW)-IOCTAL(1)                           004625
      LENGRW=LENGRW-1                                                 004630
C                                                                     004635
C     CONSTRUCT PIVOTING DATA - - - IPIV AND JPIV                     004640
C                                                                     004645
      IREG=IFLROW                                                     004650
      IREG=(SHIFT(IREG,20)) + J                                       004655
      INTFG(K+N+2)=IREG                                               004660
      GO TO 301                                                       004665
C                                                                     004670
C     FIND MOST STABILIZING PIVOT                                     004675
C                                                                     004680
  300 IHOLD=-1                                                        004685
      STRPIV=0.                                                       004690
      KKL=KAY                                                         004695
      DO 303 M=1,LENGRW                                               004700
      TEST=REALS(KKL).AND.IOCTAL(5)                                   004705
      IF(ABS(TEST).LE.ABS(STRPIV)) GO TO 400
      J=REALS(KKL).AND.IOCTAL(4)
      ISAVE=KKL
      STRPIV=TEST
  400 KKL=KKL+1
  303 CONTINUE
      PIVOT=STRPIV
      IHOLD=ISAVE
      IF(PIVOT.EQ.0.0) STOP "SINGULAR"
      GO TO 410
C
C     IF THE LAST ROW/COLUMN COORDINATES HAVE BEEN LOADED INTO
C     IPIV AND JPIV, THERE ARE NO MORE ROWS INTO WHICH WE CAN
C     SUBSTITUTE. EXIT THIS "DO LOOP".
C
  301 IF(K.EQ.N) GO TO 20
C
C-------------------------------------------------------------------
C
C     OTHERWISE, WHAT ARE THE ROWS WHICH HAVE NOT YET BEEN USED IN
C     PIVOTING AND HAVE NON-ZEROS IN THE JTH COLUMN TO BE ELIMINATED?
C
      IROWS=0
      KKK=1
      ISTRTR=K+1
          NUM=((SHIFT(INTEG(J),-40)).AND.IOCTAL(3))
          DO 2 I=ISTRTR,N
          ICHECK=INTEG(I).AND.IOCTAL(3)
          LENG=(SHIFT(INTEG(I),-20)).AND.IOCTAL(3)
C
               DO 3 L=1,LENG
               JAICK=REALS(ICHECK).AND.IOCTAL(4)
C
C     ONCE THE CHECK IS MADE FOR JAICK .GT. J, IT IS ASSUMED
C     THE VALUE WILL NOT BE FOUND.
```

```
C
               IF(JAI  .GT.J) GO TO 2
               IF(JAICK.NE.J) GO TO 30
               JCOLL=I
               JCOL=SHIFT(JCOLL,40)
               NRD=KKK+N+2
               INTEG(NRD)=(INTEG(NRD).AND.(.NOT.MASK(20)))+JCOL
               IROWS=IROWS+1
               KKK=KKK+1
               GO TO 2
  30           ICHECK=ICHECK+1
  3            CONTINUE
           IF(IROWS.GE.NUM) GO TO 27
  2         CONTINUE
C
C     REMOVE THE PIVOT ROW'S COLUMN ELEMENTS FROM CONTENTION FOR
C     FUTURF COLUMN STATUS PIVOT SELFCTION.
C
  27  IISTRT=KAY
      ISTRT=IISTRT
      DO 17 MFP=1,LENGRW
      JJJ=REALS(ISTRT).AND.IOCTAL(4)
      INTEG(JJJ)=INTEG(JJJ)-IOCTAL(2)
      ISTRT=ISTRT+1
  17  CONTINUE
C
C     SUBSTITUTE.......
C
      IF(IROWS.LE.0) GO TO 1
C
C-------------------------------------------------------------------
C
C     NOW THAT THE TALLY OF ROWS IS DONE, SUBSTITUTE THE EQUIVALENT
C     PIVOT VALUE INTO EACH ROW....COLUMN FOR COLUMN.  OBSERVE FILL-IN.
C
```

```
          DO 4 M=1,IROWS
          ISUB=(SHIFT(INTEG(N+2+M),-40)).AND.IOCTAL(3)
C
C     FETCH SUBSTITUTIONAL ELEMENT:
C
      LENGSUB=(SHIFT(INTEG(ISUB),-20)).AND.IOCTAL(3)
      KKL=INTEG(ISUB).AND.IOCTAL(3)
      KLIP=KKL
      INDEXS=KKL-1
      DO 16 M1=1,LENGSUB
      JAIN=REALS(INDEXS+M1).AND.IOCTAL(4)
      IF(J.EQ.JAIN) GO TO 160
   16 CONTINUE
      FACTOR=0.0
      GO TO 161
  160 FACTOR=REALS(INDEXS+M1).AND.IOCTAL(5)
      IHOLD=INDEXS+M1
C
C     SINCE THIS LOCATION IS BEING SUBSTITUTED FOR, DELETE IT. IT WILL
C     SOON EQUAL ZERO
C
  161 MAX=(INTEG(N+1).AND.IOCTAL(3))-2
      DO 172 M3=IHOLD,MAX
      REALS(M3)=REALS(M3+1)
  172  CONTINUE
      L1=ISUB+1
      L2=N+1
      DO 18 M3=L1,L2
      INTEG(M3)=INTEG(M3)-1
   18 CONTINUE
      INTEG(ISUB)=INTEG(ISUB)-IOCTAL(1)
      LENGSUB=LENGSUB-1
C
C     THE NUMBER OF CALCULATIONS IS THE NUMBER OF NON-ZEROS LEFT
C     EXCEPT THE PIVOT ELEMENT WHICH HAS ALREADY BEEN DELETED.
```

```
C
C-----------------------------------------------------------------------
C
C     RE-COMPUTE CONSTANT TERM.
C
          REALS(ISUB)=REALS(ISUB)-REALS(IELROW)*FACTOR
C
C     NOW WORK ON THE A MATRIX VALUES NOTING THE FILL-IN
C
          ISTRT=IISTRT
               DO 5 L=1,LENGRW
               JJ=REALS(ISTRT).AND.IOCTAL(4)
               XFER=(REALS(ISTRT).AND.IOCTAL(5))*FACTOR
C
C     SEE IF A JTH COLUMN EXISTS IN ROW "ISUB"
C
      LENGSUB=(SHIFT(INTEG(ISUB),-20)).AND.IOCTAL(3)
      KG9=KLIP
      INDEXF=KG8-1
      DO 64 M9=1,LENGSUB
      JAIN=REALS(INDEXF+M9).AND.IOCTAL(4)
      IF(JJ.EQ.JAIN) GO TO 664
   64 CONTINUE
      VALUE=0.0
      IHOLD=-1
      GO TO 604
  664 VALUE=REALS(INDEXF+M9).AND.IOCTAL(5)
      IHOLD=INDEXF+M9
  604 VALNEW=VALUE-XFER
C
C     IF THERE IS A VALUE, COMPUTE IT. IF NOT, FILL-IN.
C
      IF(IHOLD.LE.0) GO TO 200
      REALS(IHOLD)=(VALNEW.AND.IOCTAL(5)).OR.JJ
      GO TO 201
```

```
C
C     FILL-IN:
C
  200 KLIP=INTEG(ISUB).AND.IOCTAL(3)
      JAK=REALS(KLIP).AND.IOCTAL(4)
      LENGFIL=(SHIFT(INTEG(ISUB),-20)).AND.IOCTAL(3)
      JAK1=REALS(KLIP+LENGFIL-1).AND.IOCTAL(4)
      IF(JJ.LT.JAK) GO TO 50
      IF(JJ.GT.JAK1) GO TO 51
      MFIN=(INTEG(ISUB+1).AND.IOCTAL(3))-1
C
C     FIND BOUNDS ON JJ:
C
      DO 222 M4=KLIP,MFIN
      JAM=REALS(M4).AND.IOCTAL(4)
      JAM1=REALS(M4+1).AND.IOCTAL(4)
      IF((JJ.GT.JAM).AND.(JJ.LT.JAM1)) GO TO 52
  222 CONTINUE
      IAI1=INTEG(ISUB+1).AND.IOCTAL(3)
      IF(KLIP.NE.IAI1) STOP "BAD FILL-IN"
      IBUMPS=IAI1
      GO TO 60
   50 IBUMPS=KLIP
      GO TO 60
   51 IBUMPS=INTEG(ISUB+1).AND.IOCTAL(3)
      GO TO 60
   52 IBUMPS=M4+1
   60 IAN1=INTEG(N+1).AND.IOCTAL(3)
      IBMPLN=IAN1-IBUMPS
      DO 70 L4=1,IBMPLN
      INDFILL=IAN1-L4
      REALS(INDFILL+1)=REALS(INDFILL)
   70 CONTINUE
C
C     ASSIGN FILL-IN ITS NEW LOCATION:
```

```
C
      REALS(IWUMPS)=(VALNEW.AND.IOCTAL(5)).OR.JJ
C
C     RE-ARRANAGE COORDINATE TABLES
C
      INI=ISUB+1
      NIN=N+1
      DO 80 L5=INI,NIN
      INTEG(L5)=INTEG(L5)+1
   80 CONTINUE
C
C     UPDATE STATUS VECTORS:
C
      INTEG(ISUB)=INTEG(ISUB)+IOCTAL(1)
      INTEG(JJ)=INTEG(JJ)+IOCTAL(2)
      IAN1=INTEG(N+1).AND.IOCTAL(3)
      IF(IAN1.GE.IREALD) STOP "TOO MUCH FILL-IN"
  201 ISTRT=ISTRT+1
    5           CONTINUE
    4       CONTINUE
    1 CONTINUE
   20 ICHECK=(SHIFT(INTEG(2+2*N),-20)).AND.IOCTAL(3)
C
C     MAKE SURE THAT THE LAST PIVOT HAS BEEN ELIMINATED
C
      ISTT=(SHIFT(INTEG(ICHECK),-20)).AND.IOCTAL(3)
      IF(ISTT.GT.0) STOP "LAST PIVOT BAD"
      RETURN
C-----------------------------------------------------------------------
      END
      SUBROUTINE GAUSSBX(DUMMY)
C*****************************************************************
C
      COMMON INTEG(202),INTEGD,REALS(5600),IREALD,N,X(100),IOCTAL(5)
C
C     THIS SUBROUTINE BACK-SOLVES THE UPPER TRIANGULAR MATRIX OF THE
C     FORWARD GAUSS SUBROUTINE.
C                NOTE: "DUMMY" IS PASSED HERE ONLY TO SATISFY
C                      TIMER ARGUMENT NEEDS.
      DUMMI=DUMMY
C
      DO 1 K=1,N
      KK=N+1-K
      IROW=(SHIFT(INTEG(KK+N+2),-20)).AND.IOCTAL(3)
      JCOL=INTEG(KK+N+2).AND.IOCTAL(3)
      X(JCOL)=REALS(IROW)
C
C     WITH FIRST X, IT ONLY REFERENCES WITH A CONSTANT; JUMP LOOP.
C
      IF(KK.EQ.N) GO TO 1
          KX=K-1
          DO 2 L=1,KX
          JJ=N+1-L
          J=INTEG(JJ+N+2).AND.IOCTAL(3)
      KAY=INTEG(IROW).AND.IOCTAL(3)
      LENGBX=(SHIFT(INTEG(IROW),-20)).AND.IOCTAL(3)
      INDEXBX=KAY-1
           DO 20 M1=1,LENGBX
           JAIN=REALS(INDEXBX+M1).AND.IOCTAL(4)
           IF(J.EQ.JAIN) GO TO 30
   20 CONTINUE
      GO TO 2
   30 VALUE=(REALS(INDEXBX+M1)).AND.IOCTAL(5)
C
C     ITERATE X WITH PRODUCTS OF ELEMENTS AND PREVIOUSLY SOLVED FOR
```

```
C     X VALUES AS DENOTED BY JPIV.
C
          X(JCOL)=X(JCOL)-VALUE*X(J)
    2     CONTINUE
    1 CONTINUF
      RETURN
C
C     NOTFS.....THE PIVOT VALUE IS 1., SO IT HAS BEEN PREVIOUSLY
C     DELETED. SINCE A IS UPPER TRIANGULAR, IT IS IMPOSSIBLE TO HAVE TO
C     MULTIPLY AN A(I,J) BY AN X(J) WHICH HAS NOT YET BEEN CALCULATED.
C
C     THE MATRIX IS NOT STORED AS UPPER TRIANGULAR, BUT IPIV AND
C     JPIV ORDER THE MANIPULATIONS AS IF IT WERE UPPER TRIANGULAR.
C     RECALL THAT IPIV AND JPIV ARE STORED AS BITS OF INTEG(N+3) ON.
C---------------------------------------------------------------------
      END
```

Vita

Michael Francis Poore was born on 3 February 1949 in Washington, D.C. He graduated from High School in Bethesda, Maryland in 1967 and attended the University of Notre Dame from which he received the degree of Bachelor of Science, Electrical Engineering in May 1971. Upon graduation, he received a commission in the Regular USAF through the AFROTC program, after which he entered USAF Undergraduate Pilot Training at Sheppard AFB, Texas. He achieved the aeronautical rating of Pilot in June 1972. He then served as an aircraft commander in the RF-4C in the 91st Tactical Reconnaissance Squadron, 67th Tactical Reconnaissance Wing, at Bergstrom AFB, Texas. In June 1975, he entered the Graduate Electro-Optics Program of the School of Engineering, Air Force Institute of Technology.

Permanent Address: 8200 Research Blvd.
Austin, Texas 78758

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
GEO/PH/77-3 ✓

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
DIRECT SOLUTIONS OF LARGE,
SPARSE LINEAR SYSTEMS

5. TYPE OF REPORT & PERIOD COVERED
M.S. Thesis

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Michael F. Poore
Captain, USAF

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS ✓
Air Force Institute of Technology(AFIT/EN)
Wright-Patterson AFB, Ohio 45433

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Project 2304-Y1-0I

11. CONTROLLING OFFICE NAME AND ADDRESS
Air Force Weapons Laboratory (AFWL/DYM)
Kirtland AFB, New Mexico 87117

12. REPORT DATE
December 1977

13. NUMBER OF PAGES
150

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES
Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Matrix
Sparse Matrix
Linear Systems
Linear Algebra
Gaussian Elimination

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A comparison is made of the merits of three popular algorithms for direct solutions of large, sparse matrices: Gaussian Elimination, LU Decomposition, and Gauss-Jordan Reduction. The last two algorithms are used in existing sparse matrix solvers at the Air Force Weapons Lab, Kirtland AFB, NM. A mathematical theory discussion explains the algorithms and predicts their performance for arbitrary and strongly structured matrices.

Insert Page

The performance comparison involves a wide range of problems practical to technical study at the Weapons Lab. Particular emphasis is placed on solution accuracy and the efficient use of core space. The same test problems are used to analyze the Gaussian Elimination algorithm programmed by this student. From a study of the performance of several Gaussian solution strategies, a new strategy is developed which offers the user a range of options for his particular programming needs. The salient points of this strategy include some stability features of partial pivoting and some array optimization similar to minimum row/minimum column pivoting. The final Gaussian Elimination program is enhanced by a new packing scheme which is highly suited for the CDC 6600 computer: many arrays can be compacted into a single array by subdividing the long computer word structure. A final qualitative comparison is presented from which an optimal solution method is proposed and further study recommended.